

Helmut Herold

Linux · Unix Shells

Bourne-Shell, Korn-Shell,
C-Shell, bash, tcsh



ADDISON-WESLEY



LINUX-UNIX-Shells

LINUX / UNIX und seine Werkzeuge

bisher erschienen:

Helmut Herold: LINUX-UNIX-Kurzreferenz

Helmut Herold: LINUX-UNIX-Grundlagen

Helmut Herold: LINUX-UNIX-Profitools

Helmut Herold: LINUX-UNIX-Shells

Helmut Herold: LINUX-UNIX-Systemprogrammierung

Helmut Herold

LINUX-UNIX-Shells

Bourne-Shell, Korn-Shell, C-Shell, bash, tcsh

3., aktualisierte Auflage



ADDISON-WESLEY

An imprint of Pearson Education Deutschland GmbH

München • Bosten • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Herold, Helmut:

Linux-Unix-Shells : Bourne-Shell, Korn-Shell, bash, tcsh /

Helmut Herold. – 3. Aufl. – Bonn ; Reading, Mass. [u. a.] :

Addison-Wesley-Longman, 1999.

(Linux / Unix und seine Werkzeuge)

ISBN 3-8273-1511-5

Buch .1999

GB

10 9 8 7 6 5 4 3 2
03 02 01 00 99

© 1999 by Addison Wesley Longman Verlag, ein Imprint der Pearson Education Deutschland GmbH

Lektorat: Susanne Spitzer, München

Satz: Reemers EDV-Satz, Krefeld. Gesetzt aus der Palatino 9,5 Punkt

Belichtung, Druck und Bindung: Kösel GmbH, Kempten

Produktion: TYPisch Müller, München

Umschlaggestaltung: Hammer Grafik-Design, Haar bei München

Das verwendete Papier ist aus chlorfrei gebleichten Rohstoffen hergestellt und alterungsbeständig. Die Produktion erfolgt mit Hilfe umweltschonender Technologien und unter strengsten Auflagen in einem geschlossenen Wasserkreislauf unter Wiederverwertung unbedruckter, zurückgeführter Papiere.

Text, Abbildungen und Programme wurden mit größter Sorgfalt erarbeitet. Verlag, Übersetzer und Autoren können jedoch für eventuell verbliebene fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Kein Teil dieses Buches darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form durch Fotokopie, Mikrofilm oder andere Verfahren reproduziert oder in eine für Maschinen, insbesondere Datenverarbeitungsanlagen, verwendbare Sprache übertragen werden. Auch die Rechte der Wiedergabe durch Vortrag, Funk und Fernsehen sind vorbehalten.

Die in diesem Buch erwähnten Software- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Warenzeichen und unterliegen als solche den gesetzlichen Bestimmungen.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Übersicht zu diesem Buch	1
1.2	Hinweis zum Herunterladen der im Buch vorgestellten Shellskripts	2
1.3	Hinweis zur Buchreihe: Linux/ Unix und seine Werkzeuge	3
2	Allgemeines zur UNIX-Shell	5
2.1	Aufgaben einer Shell	5
2.2	Shell-Varianten	6
2.3	Eigenschaften der Shell	7
3	Begriffe der Shell	9
4	Die Bourne-Shell	11
4.1	Metazeichen	11
4.2	Einfache Kommandos, Pipelines und Listen	13
4.3	Kommentare	21
4.4	Shell-Skripts (Shell-Prozeduren)	22
4.5	Kommandosubstitution	27
4.6	Shell-Parameter	31
4.7	Expandierung von Dateinamen auf der Kommandozeile	64
4.8	Quoting	67
4.9	Ein- und Ausgabeumlenkung	78
4.10	Auswertung von Ausdrücken	92
4.11	Kommandoklammerung	102
4.12	Programmiersprachkonstrukte der Shell	107
4.13	Fehlersuche in Shell-Skripts	168
4.14	Signalbehandlung in der Shell	180
4.15	Builtin-Kommandos der Bourne-Shell	188
4.16	Abarbeitung von Kommandozeilen	211
4.17	Die Aufrufsyntax der Shell	216
4.18	Handhabung von überlangen Kommandozeilen	219

4.19	Die Datei .profile	223
4.20	Die eingeschränkte Shell rsh	227
4.21	Job-Kontrolle mit shl	229
4.22	Anwendungsbeispiele	234
5	Die Korn-Shell	249
5.1	Erweiterungen und Neuheiten der Korn-Shell	249
5.2	Das Starten der Korn-Shell	251
5.3	Metazeichen	253
5.4	Einfache Kommandos, Pipelines und Listen	255
5.5	Kommentare	257
5.6	Shell-Skripts (Shell-Prozeduren)	257
5.7	Kommandosubstitution	258
5.8	Shell-Parameter	261
5.9	Expandierung von Dateinamen auf der Kommandozeile	287
5.10	Quoting	290
5.11	Ein- und Ausgabeumlenkung	293
5.12	Die builtin-Kommandos read und print	301
5.13	Überprüfen von Bedingungen	306
5.14	Arithmetische Auswertungen	310
5.15	Kommandoklammerung	318
5.16	Programmiersprachkonstrukte der Korn-Shell	319
5.17	Der Alias-Mechanismus	343
5.18	Die Tilde-Expandierung	353
5.19	Der History-Mechanismus	353
5.20	Die builtin-Editoren	359
5.21	Fehlersuche in ksh-Skripts	385
5.22	Signalbehandlung in der ksh	387
5.23	Job-Kontrolle in der Korn-Shell	391
5.24	Builtin-Kommandos der Korn-Shell	400
5.25	Die Abarbeitung von Kommandozeilen	416
5.26	Aufrufsyntax und Umgebung der ksh	428
5.27	Einrichten einer persönlichen Arbeitsumgebung	437
5.28	Anwendungsbeispiele	447
5.29	Unterschiede zwischen der Bourne-Shell und ksh-Versionen	468
5.30	Literaturhinweis	476

6 Die C-Shell	477
6.1 Starten und Beenden der C-Shell	478
6.2 Metazeichen	479
6.3 Einfache Kommandos, Pipelines und Listen	482
6.4 Kommentare	484
6.5 Shell-Skripts (Shell-Prozeduren)	485
6.6 Kommandosubstitution	485
6.7 Shell-Parameter	486
6.8 Expandierung von Dateinamen auf der Kommandozeile	501
6.9 Quoting	504
6.10 Ein- / Ausgabeumlenkung	506
6.11 Ausdrücke	509
6.12 Kommandoklammerung mit (..)	517
6.13 Kommandos zur Ablaufsteuerung	517
6.14 Der History-Mechanismus	528
6.15 Der Alias-Mechanismus	535
6.16 Fehlersuche in csh-Skripts	539
6.17 Signalbehandlung in der csh	540
6.18 Job-Kontrolle in der csh	542
6.19 Builtin-Kommandos der C-Shell	552
6.20 Die Abarbeitung von Kommandozeilen	567
6.21 Aufrufsyntax der csh	568
6.22 Einrichten einer persönlichen Arbeitsumgebung	570
6.23 Erweiterungen in manchen csh-Versionen	574
6.24 Anwendungsbeispiel	575
7 bash – Bourne Again Shell	585
7.1 Wesentliche Unterschiede der bash zur Bourne-Shell	585
7.2 Optionen und Argumente für die bash	588
7.3 Einfache Kommandos, Pipelines und Listen	589
7.4 Kommandos zur Ablaufsteuerung	589
7.5 Kommentare	591
7.6 Quoting	591
7.7 Parameter und Variablen	591
7.8 Substitutionen und Expandierungen	596
7.9 Ein- / Ausgabeumlenkung	600
7.10 Funktionen	601
7.11 Alias-Mechanismus	602

7.12	Jobkontrolle	602
7.13	Der bash-Prompt	603
7.14	Editieren von Kommandozeilen	604
7.15	History-Mechanismus	613
7.16	bash-Arithmetik	615
7.17	Builtin-Kommandos	617
7.18	Startup-Dateien der bash	633
7.19	Abarbeitung von Kommandozeilen	634
7.20	Anwendungsbeispiel	635
8	tcsh – Erweiterte C-Shell	639
8.1	Aufrufsyntax der tcsh	639
8.2	Startup-Dateien der tcsh	640
8.3	Editieren von Kommandozeilen	641
8.4	Lexikalische Struktur	658
8.5	Substitutionen	659
8.6	Ausführung von Kommandos	670
8.7	tcsh-Ausdrücke	673
8.8	Jobkontrolle	677
8.9	Periodische, automatische und zeitgesteuerte Ereignisse	678
8.10	Abfangen von Signalen	679
8.11	Builtin-Kommandos	679
8.12	Spezielle Aliase	706
8.13	Spezielle Shellvariablen	707
8.14	Environment	725
8.15	Anwendungsbeispiel	726
9	Die beiden Tischrechner dc und bc	729
9.1	Der Tischrechner dc	729
9.2	Der Tischrechner bc	738
10	Dialogboxen unter Linux	761
10.1	Aufrufsyntax des dialog-Kommandos	761
10.2	Ausgeben von Informationen (--msgbox, --infobox, --textbox)	762
10.3	Ja / Nein-Auswahl (--yesno)	765
10.4	Eingabe von Strings (--inputbox)	768
10.5	Menüauswahl (--menu)	771
10.6	Auswahl von Optionen (--checklist)	774
10.7	Auswahl einer von mehreren Alternativen (--radiolist)	781

10.8	Ausgabe von Prozentbalken (--gauge)	784
10.9	Konfiguration von dialog-Ausgaben	788
10.10	Anwendungsbeispiel	790
 Stichwortverzeichnis		 797

1 Einleitung

*Dimidum facti, qui coepit, habet.
(Wer nur begann, der hat schon halb vollendet.)*

Horaz

Die UNIX-Shell ist zunächst einmal der UNIX-Kommandointerpreter, welcher die vom Benutzer eingegebenen Kommandos liest und nach einigen Aufbereitungen zur Ausführung bringt. Neben dieser wichtigen Aufgabe der Kommandointerpretation verfügt die UNIX-Shell auch über Fähigkeiten, die von höheren Programmiersprachen bekannt sind wie Verzweigungen, Schleifen und Funktionsdefinitionen. Dies macht eine Shell zu einem mächtigen Werkzeug, ohne das kein Software-Entwickler unter UNIX auskommt.

Unglücklicherweise existiert nicht nur eine einzige Shell, sondern es wurden auf den verschiedenen UNIX-Derivaten und -Versionen eigene Shells entwickelt. Dieses Buch stellt die gebräuchlichsten UNIX-Shells vor: **Bourne-Shell**, **Korn-Shell**, **C-Shell**, **Bourne-Again-Shell** und **TC-Shell**.

Da die ursprüngliche Shell, die Bourne-Shell, über keine eingebaute Arithmetik verfügt, müssen bei Berechnungen in dieser Shell andere UNIX-Kommandos mit Rechenfähigkeiten aufgerufen werden. Zwei Kommandos, die solche Fähigkeiten aufweisen, werden neben den fünf Shells in einem eigenen Kapitel vorgestellt. Es handelt sich dabei um die beiden **Tischrechner bc und dc**.

1.1 Übersicht zu diesem Buch

Kapitel 2 dieses Buches beschäftigt sich mit der UNIX-Shell im allgemeinen. Dazu beschreibt es neben den Aufgaben und Eigenschaften der Shell noch kurz die fünf hier vorgestellten Shell-Varianten.

Kapitel 3 stellt allgemeine Shell-Begriffe vor, welche in allen Shell-Varianten verwendet werden.

Kapitel 4 ist ein sehr umfangreiches Kapitel, das die Bourne-Shell vorstellt. Da die Bourne-Shell die erste Shell überhaupt war, stellt sie den Ausgangspunkt für alle anderen Shell-Arten dar. Dies ist auch der Grund, warum die Bourne-Shell hier äußerst detailliert beschrieben wird, denn das Verständnis der grundlegenden Konzepte der Bourne-Shell erleichtert einen Einstieg bzw. Umstieg auf andere Shells doch wesentlich.

Kapitel 5 stellt die Korn-Shell vor. Die Korn-Shell ist der natürliche Nachfolger der Bourne-Shell. Da die Korn-Shell weitgehend aufwärtskompatibel zur Bourne-Shell ist, werden in diesem Kapitel nur die Neuheiten und Unterschiede der Korn-Shell zur Bourne-Shell vertieft vorgestellt.

Kapitel 6 behandelt die C-Shell. Die C-Shell wurde auf der starken BSD-UNIX-Linie entwickelt und erfreut sich großer Beliebtheit, da ihre Syntax an die Programmiersprache C angelehnt ist und sie viele Vorzüge gegenüber der Bourne-Shell aufweist. Die Vorstellung der C-Shell beschränkt sich dabei vor allen Dingen auf die Syntax, da die grundlegenden Konzepte einer Shell bereits in den vorherigen Kapiteln ausführlich behandelt wurden.

Kapitel 7 stellt die immer beliebter werdende Bourne-Again-Shell, kurz **bash**, vor. Die bash ist ein Nachfolger der Bourne-Shell und als freie Software verfügbar. Unter Linux, das sich immer größerer Beliebtheit erfreut, ist die bash standardmäßig als Benutzer-Shell eingerichtet. In diesem Kapitel werden vor allen Dingen die Unterschiede und Neuheiten der bash gegenüber der Bourne-Shell hervorgehoben.

Kapitel 8 behandelt den Nachfolger der C-Shell, die **tcsh**. Die tcsh verhält sich genauso wie die C-Shell, bietet aber eine Vielzahl von interessanten Erweiterungen und Neuheiten, die in diesem Kapitel besonders hervorgehoben werden. Grundlegende Konzepte, die völlig identisch zur C-Shell sind, werden dabei nur kurz angesprochen.

Kapitel 9 stellt die beiden Tischrechner **bc** und **dc** vor, welche Berechnungen mit sehr großen Zahlen und größter Genauigkeit zulassen. Neben ihrer direkteren Benutzung werden diese beiden Tischrechner oft für arithmetische Berechnungen in der Bourne-Shell verwendet, da diese über keine eigene Arithmetik verfügt.

Kapitel 10 stellt dann noch das unter Linux verfügbare Kommando **dialog** vor, das die Darstellung von farbigen Dialogboxen aus Shell-Skripts heraus erlaubt. Mit dem dialog-Kommando ist das Erstellen benutzerfreundlicher und ansprechender Oberflächen zu Shell-Skripts sehr leicht.

1.2 Hinweis zum Herunterladen der im Buch vorgestellten Shellskripts

Alle im Buch vorgestellten Shellskripts befinden sich als gezippte tar-Datei **shells.tgz** auf der WWW-Seite <http://www.addison-wesley.de/service/herold> und können heruntergeladen werden. Die Shellskripts der jeweiligen Kapitel befinden sich in folgenden Sub-Directories in dieser gezippten tar-Datei:

shellueb	Kapitel 4 (Die Bourne-Shell)
kshueb	Kapitel 5 (Die Korn-Shell) Sub-Directory kshueb/fl1b enthält die Funktionsdefinitions-Dateien zu diesem Kapitel
cshueb	Kapitel 6 (Die C-Shell)
bashueb	Kapitel 7 (bash – Die Bourne-Again-Shell)
tcshueb	Kapitel 8 (tcsh – Die erweiterte C-Shell)
bc dc	Kapitel 9 (Die beiden Tischrechner dc und bc)
dialog	Kapitel 10 (Dialogboxen unter Linux)

1.3 Hinweis zur Buchreihe: LINUX/UNIX und seine Werkzeuge

Diese Buchreihe soll

- ▶ den Linux/Unix-Anfänger systematisch vom Linux/Unix-Basiswissen, über die mächtigen Linux/Unix-Werkzeuge bis hin zu den fortgeschrittenen Techniken der Systemprogrammierung führen.
- ▶ dem bereits erfahrenen Linux/Unix-Anwender – aufgrund ihres modularen Aufbaus – eine Vertiefung bzw. Ergänzung seines Linux/Unix-Wissens ermöglichen.

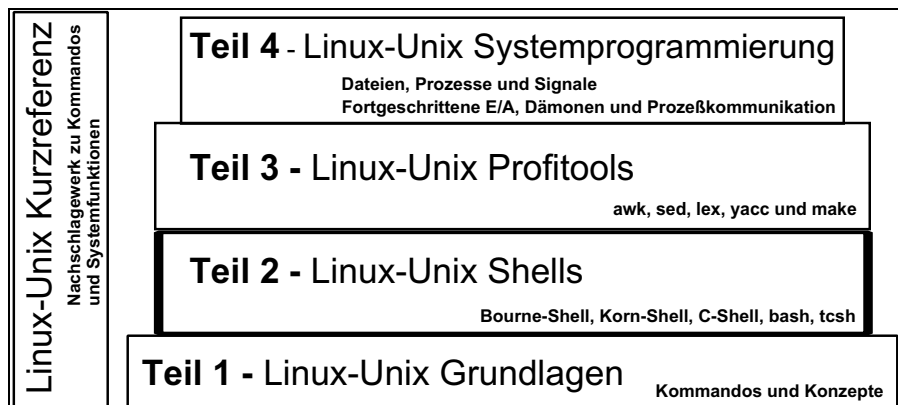


Abbildung 1.1: Die Buchreihe »Linux/Unix und seine Werkzeuge«

2 Allgemeines zur UNIX-Shell

Magna pars est profectus velle proficere.
(Ein großer Teil des Fortschreitens besteht darin, daß wir fortschreiten wollen.)

Seneca

2.1 Aufgaben einer Shell

Ein Benutzer kann normalerweise nicht direkt mit der Hardware eines Rechners oder mit der über der Hardware angeordneten Schicht, dem Betriebssystemkern, kommunizieren. Da eine solche Kommunikation viel zu komplex und benutzerunfreundlich wäre, wurde zwischen diese tieferliegenden Ebenen und die Benutzerebene eine weitere Schicht eingeschoben. Diese Schicht nimmt die Kommandos des Benutzers entgegen, interpretiert diese und setzt sie in Systemaufrufe um, so daß die vom Benutzer geforderten Aktivitäten vom System durchgeführt werden. Die System-Rückmeldungen werden dann wieder von dieser Schicht entgegengenommen und ausgewertet; im Fehlerfalle werden sie dann in entsprechende Meldungen an den Benutzer umgesetzt. Diese Zwischenschicht trägt unter UNIX den Namen *Shell* (deutsch: Schale, Muschel), weil sie sich wie eine Schale um den Betriebssystemkern legt (siehe Abbildung 2.1).

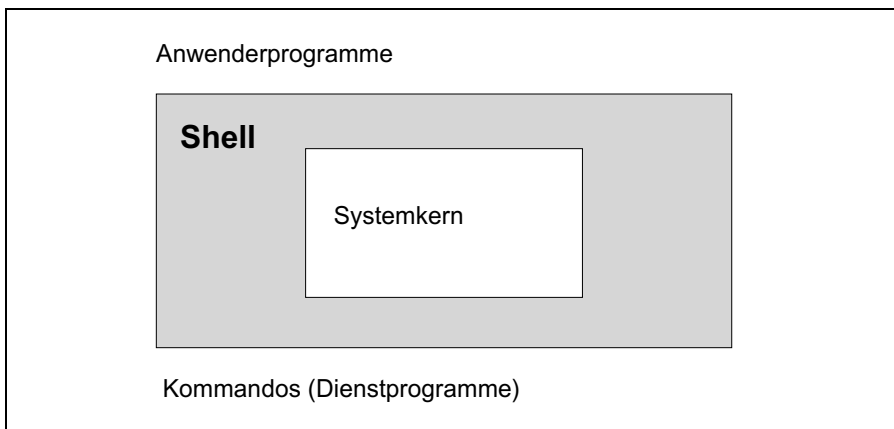


Abbildung 2.1: Die Shell als Mittler zwischen Betriebssystemkern und Benutzer

Im Unterschied zu vielen anderen Systemen, in denen diese Schicht oft als Monitor bezeichnet wird, ist die UNIX-Shell kein Bestandteil des Betriebssystemkerns, sondern ein eigenes Programm, das sich zwar bezüglich der Leistungsfähigkeit von anderen UNIX-Kommandos erheblich unterscheidet, aber doch wie jedes andere UNIX-Kommando oder Anwenderprogramm aufgerufen oder sogar ausgetauscht werden kann.

2.2 Shell-Varianten

Da die Shell einfach austauschbar ist, wurden von einigen UNIX-Linien bzw. UNIX-Versionen eigene Shell-Varianten entwickelt. Hier werden die am weitesten verbreiteten UNIX-Shells behandelt:

Bourne-Shell

Diese Shell ist nach ihrem Erfinder Steve Bourne benannt, und war die erste UNIX-Shell. Sie wird deshalb auch als die Standard-Shell bezeichnet.

Korn-Shell

Bei dieser Shell, die nach ihrem Erfinder David Korn benannt ist, handelt es sich um eine verbesserte Version der Bourne-Shell. Die Korn-Shell ist weitgehend aufwärtskompatibel zur Bourne-Shell. Das bedeutet, daß die Korn-Shell über alle Funktionen der Bourne-Shell verfügt, aber noch zusätzliche neue Mechanismen anbietet wie z.B. einen History-Mechanismus, der das Editieren und die erneute Ausführung früher gegebener Kommandos zuläßt, ohne daß diese vollständig neu einzugeben sind.

C-Shell

Diese Shell wurde auf der BSD-UNIX¹ Linie entwickelt. Sie basiert zwar auf dem Konzept der Bourne-Shell, ihre Syntax ist jedoch sehr stark an die Programmiersprache C angelehnt (daher der Name C-Shell). Auch die C-Shell bietet nützliche Funktionen an, über welche die Bourne-Shell nicht verfügt, wie z.B. Job-Kontrolle oder einen History-Mechanismus.

Bourne-Again-Shell (bash)

Die Bourne-Again-Shell (bash) ist ein Nachfolger der Bourne-Shell und als freie Software verfügbar. Unter Linux, das sich immer größerer Beliebtheit erfreut, ist sie standardmäßig als Benutzer-Shell eingerichtet. Die bash enthält Elemente aus allen anderen hier vorgestellten Shells, wobei sie jedoch auch eigene neue Konstrukte enthält. Die wohl erfreulichste Neuheit gegenüber der Bourne-Shell ist der Zugriff auf früher eingegebene Kommandozeilen und das Editieren von

1. BSD = **B**erkeley **S**ystem **D**istribution

Kommandozeilen, was auch mit der Grund sein dürfte, daß immer mehr Anwender von der Bourne-Shell auf die bash umsteigen.

TC-Shell (tcsh)

Die tcsh ist ein Nachfolger der C-Shell mit zahlreichen interessanten Erweiterungen und Neuheiten. Da sie zur C-Shell weitgehend abwärtskompatibel ist, ist ein Umstieg von der C-Shell auf die tcsh nicht allzu schwierig. So ist z.B. in der tcsh der Zugriff auf vorherige Kommandozeilen oder das Editieren der aktuellen Kommandozeile wesentlich einfacher als in der C-Shell. Deshalb steigen auch immer mehr C-Shell-Benutzer von der C-Shell auf die tcsh um.

Welche dieser Shells als Login-Shell nach dem Anmelden eines Benutzers zu starten ist, kann der Systemadministrator durch einen Eintrag in der entsprechenden Benutzerzeile in der Datei */etc/passwd* festlegen. Kein Eintrag entspricht dabei der Angabe: **/bin/sh**. Eine Zeile in der Paßwort-Datei */etc/passwd* bezieht sich jeweils auf einen Benutzer. Innerhalb jeder Zeile sind die einzelnen Felder durch Doppelpunkte getrennt und in folgender Reihenfolge angeordnet:

```

heh:hu671sKjhg8ah:118:109:Helmut Herold:/user1/heh:/bin/sh  (Bourne-Shell)
ali:h269usHGp253f:143:111:Albert Igel:/user1/ali:          (Bourne-Shell)
fme:aF9Zhhksdji9a:121:110:Fritz Meyer:/user2/fme:/bin/ksh   (Korn-Shell)
mik:UijKp673ha7ui:138:110:Michael Kode:/user2/mik:/bin/csh (C-Shell)

```

| | | | | | |
 | | | | | | | Login-Shell
 | | | | | | | home directory
 | | | | | | | Weitere Info. zum Benutzer
 | | | | | | | (oft: richtiger Name)
 | | | | | | | Gruppennummer (GID)
 | | | | | | | Benutzernummer (UID)
 | | | | | | | Verschlüsseltes Paßwort
 | | | | | | | Login-Kennung

2.3 Eigenschaften der Shell

Die wichtigste Aufgabe der Shell ist die *Kommandointerpretation*: Nach der Ausgabe des Promptzeichens \$¹ durch die Login-Shell kann der Benutzer Kommandos eingeben. Das eingegebene Kommando wird dann nicht – wie bereits erwähnt – vom Betriebssystemkern, sondern von der Shell entgegengenommen, die dann – nach einigen Aufbereitungen (wie z.B. Expandierung von Dateinamen²) – durch entsprechende Systemaufrufe für die Ausführung des Kommandos sorgt.

1. Im Falle des Superusers wird das Promptzeichen # verwendet.

2. Siehe Kapitel 2.6 »Expandierung von Dateinamen auf der Kommandozeile« in »Linux/Unix-Grundlagen«.

Ein oder mehrere Kommandos können auch in eine Datei geschrieben werden. Der Aufruf einer solchen Datei bewirkt dann, daß die darin angegebenen Kommandos der Reihe nach von der Shell gelesen und zur Ausführung gebracht werden. Solche Kommandodateien werden auch Shell-Skripts (oder Shell-Prozeduren) genannt.

Shell-Skripts müssen nicht kompiliert werden, die Shell arbeitet die darin angegebenen Kommandos wie ein Interpreter Schritt für Schritt ab.

Neben dieser wichtigen Aufgabe der Kommandointerpretation verfügt die Shell über einige *Programmiersprach-Fähigkeiten*: Shell-Variablen, Ablaufstrukturen wie **if**- und **case**-Anweisungen, **for**- und **while**-Schleifen und die Definition von Unterprogrammen.

Die Shell ist ein sehr mächtiges und vielseitiges Werkzeug, das mit seinen knappen und oft auch kryptisch wirkenden Konstrukten besonders auf die Bedürfnisse von Software-Entwicklern zugeschnitten ist. Dies hat leider auch zur Folge, daß viel Übung und Erfahrung notwendig ist, um ihre gesamten Möglichkeiten anzuwenden. Der erste einfache Umgang mit der Shell ist dennoch leicht zu erlernen.

3 Begriffe der Shell

*Non obtinebis, ut desinat, si incipere permiseris.
(Du wirst nicht erreichen, daß einer aufhört, wenn du einmal den Anfang gestattet hast.)*

Seneca

Kommandozeile (*command line*)

ist eine Eingabezeile, die von der Shell zu interpretieren ist.

Metazeichen (*metacharacter*)

sind Zeichen (keine Ziffern oder Buchstaben), denen von der entsprechenden Shell eine Sonderbedeutung zugeordnet ist. Ein Beispiel für ein Metazeichen ist `>`, das die Umlenkung der Standardausgabe in eine Datei bewirkt.

Trennzeichen (*blank*)

sind Leer- oder Tabulatorzeichen.

Bezeichner (*identifier*)

ist eine Folge von Buchstaben (keine Umlaute oder ß), Ziffern oder Unterstrichen, wobei diese Folge mit einem Buchstaben oder einem Unterstrich beginnen muß.

Erlaubte Bezeichner:

Amen
ANZAHL
ZEICH_ZAEHL
A2
null_08
_MAX_ZAHL

Nicht erlaubte Bezeichner:

vier_*_hotel
2A
7_und_40_elf
kündigung (Umlaute sind nicht erlaubt)

Bezeichner werden z.B. für Namen von Shell-Variablen oder Shell-Funktionen verwendet.

Wort (*word*)

ist eine Folge von Zeichen, die durch ein oder mehrere der folgenden Metazeichen

; & () | < > *Neuezeile-Zeichen Leerzeichen Tabulatorzeichen*

abgegrenzt ist; mit \ ausgeschaltete Metazeichen (wie z.B. \;) sind allerdings Teil eines Wortes.

Die Kommandozeile

```
ls|wc>datzahl
```

enthält drei Wörter: `ls`, `wc` und `datzahl`

Die Kommandozeile

```
cat gut\&besser|nl>numeriert
```

enthält vier Wörter: `cat`, `gut\&besser`, `nl` und `numeriert`.

4 Die Bourne-Shell

*Es genügt nicht, zum Fluß zu kommen mit dem Wunsche, Fische zu fangen.
Man muß auch das Netz mitbringen.*

Sprichwort aus China

Die Bourne-Shell war die erste UNIX-Shell überhaupt und ist auch heute noch die Standard-Shell von UNIX-System V. Neben ihrer primären Aufgabe, der Entgegennahme und Interpretation von interaktiv eingegebenen Kommandos, stellt sie auch eine vollständige Kommandosprache mit Variablen, Funktionsdefinitionen, Ablaufstrukturen, wie Verzweigungen oder Schleifen, und Signal-Handling zur Verfügung.

In den folgenden Beispielen wird das Home-Directory `/user1/egon` und das Working-Directory `/user1/egon/shellueb` angenommen.

4.1 Metazeichen

In der Bourne-Shell existiert eine Vielzahl von Metazeichen. Dies sind Zeichen, denen eine Sonderbedeutung zugeordnet ist. Tabelle 4.1 enthält nicht nur eine vollständige Übersicht dieser Metazeichen, sondern gibt auch eine Kurzbeschreibung ihrer Sonderbedeutung:

Metazeichen	Bedeutung
<code>kdo>datei</code>	Standardausgabe von <code>kdo</code> in <code>datei</code> umlenken.
<code>kdo>>datei</code>	Standardausgabe von <code>kdo</code> am Ende von <code>datei</code> anfügen.
<code>kdo<datei</code>	Standardeingabe von <code>kdo</code> auf <code>datei</code> umlenken.
<code>kdo<<wort</code>	Hier-Dokument (engl.: <i>here document</i>): Standardeingabe für <code>kdo</code> besteht aus den nächsten Zeilen bis zur ersten Zeile, die nur <code>wort</code> enthält.
<code>kdo1 kdo2</code>	Standardausgabe von <code>kdo1</code> über eine Pipe in die Standardeingabe von <code>kdo2</code> weiterleiten.
<code>*</code>	steht für »kein, ein oder mehrere Zeichen«.
<code>?</code>	steht für »ein beliebiges Zeichen«.
<code>[...]</code>	steht für »eines der in [...] angegebenen Zeichen«; Bereichsangaben wie 0-9 oder a-z sind innerhalb von [...] erlaubt.

Tabelle 4.1: Metazeichen der Bourne-Shell

Metazeichen	Bedeutung
[!...]	steht für »eines der nicht in [!...] angegebenen Zeichen«; Bereichsangaben wie 0-9 oder a-z sind innerhalb von [!...] erlaubt.
<i>kdo1</i> ; <i>kdo2</i>	Semikolon trennt mehrere Kommandos in einer Kommandozeile; nach Beendigung von <i>kdo1</i> wird <i>kdo2</i> ausgeführt.
<i>kdo</i> &	<i>kdo</i> im Hintergrund (parallel) ablaufen lassen; in diesem Fall wartet die Shell nicht auf die Beendigung von <i>kdo</i> .
` <i>kdo</i> `	Kommandosubstitution: führt <i>kdo</i> aus und ersetzt dann in der Kommandozeile ` <i>kdo</i> ` durch die Standardausgabe von <i>kdo</i> .
(<i>kdo</i>)	<i>kdo</i> in einer Subshell ausführen.
{ <i>kdo</i> ; } ^a	<i>kdo</i> in der aktuellen Shell (nicht in einer Subshell) ausführen.
\$0, \$1,..., \$9	Werte der Positionsparameter für ein Shell-Skript bzw. für eine Shell-Funktion.
\$ <i>var</i>	Wert der Shell-Variablen <i>var</i> .
\${ <i>var</i> } <i>text</i>	Wert der Shell-Variablen <i>var</i> mit nachfolgendem <i>text</i> zu einer Zeichenkette zusammenfügen.
\c	Sonderbedeutung des nachfolgenden Metazeichens <i>c</i> ausschalten.
'...'	Sonderbedeutung aller in '...' angegebenen Zeichen (außer Apostroph ') wird ausgeschaltet.
"..."	Sonderbedeutung aller in "..." angegebenen Zeichen, außer \$, `...` und \ wird ausgeschaltet.
#	leitet Kommentar ein; Rest der Zeile wird von der Shell ignoriert.
<i>var</i> = <i>wert</i>	Zuweisung von <i>wert</i> an die Shell-Variable <i>var</i> .
<i>kdo1</i> && <i>kdo2</i>	<i>kdo2</i> wird nur dann ausgeführt, wenn Ausführung von <i>kdo1</i> erfolgreich verlief.
<i>kdo1</i> <i>kdo2</i>	<i>kdo2</i> wird nur dann ausgeführt, wenn Ausführung von <i>kdo1</i> nicht erfolgreich verlief.
Neuzeile-Zeichen	übergibt eine Kommandozeile an die Shell zur Abarbeitung.
Leerzeichen	Trennzeichen für Wörter (sonst keine Sonderbedeutung).
Tabulatorzeichen	

Tabelle 4.1: Metazeichen der Bourne-Shell (Fortsetzung)

a. { und } sind eigentlich keine Metazeichen, sondern Schlüsselwörter der Shell.

Manche dieser Metazeichen wurden bereits im ersten Buch vorgestellt. Eine ausführliche Beschreibung der einzelnen Metazeichen erfolgt im Laufe dieses Kapitels. Dabei mag es teilweise zu Wiederholungen aus dem ersten Buch kommen.

Eine erneute Wiedergabe eines Konstrukts ist dabei nicht versehentlich, sondern aus folgenden Gründen beabsichtigt:

1. Dem Leser soll verdeutlicht werden, daß es sich bei den entsprechenden Metazeichen nicht um ein UNIX-Metazeichen, sondern um ein Metazeichen für das Shell-Programm handelt. Diese Unterscheidung konnte im ersten Buch aus didaktischen Gründen noch nicht ausreichend hervorgehoben werden.
2. Es ist die Intention dieser Buchreihe, durch ständige Rückgriffe auf bereits vorgestellte Konstruktionen oder Kommandos dem Leser ein bleibendes Linux/Unix-Wissen zu vermitteln.
3. Nicht alle Konstrukte wurden im ersten Buch ausführlich beschrieben, da zu diesem Zeitpunkt die Priorität auf ein schnellstmögliches Verständnis und den Überblick gelegt wurde. Die vollständige Darstellung mit allen syntaktischen und semantischen Gesichtspunkten hätte diesem Vorgehen entgegen gewirkt. Dies wird jetzt nachgeholt.

4.2 Einfache Kommandos, Pipelines und Listen

Listen werden aus einer oder mehreren *Pipelines* gebildet. *Pipelines* wiederum setzen sich aus einem oder mehreren Kommandos zusammen. *Kommandos* sind unterteilt in

- ▶ einfache Kommandos,
- ▶ geklammerte Kommandos und
- ▶ Kommandos, die an Programmiersprach-Konstrukte erinnern.

Abbildung 4.1 veranschaulicht diese Zusammenhänge.

Hinweis

Kommandoklammerung und Kommandos, die Ablaufstrukturen nachbilden, werden in den Kapiteln 4.11 und 4.12 ausführlich behandelt: Die vollständige Definition für ein Kommando ist:

- ▶ entweder ein *einfaches Kommando*
- ▶ oder eines der folgenden Konstrukte:
 - ▶ **if**-Anweisung
 - ▶ **case**-Anweisung
 - ▶ **for**-Anweisung
 - ▶ **while**-Anweisung
 - ▶ **until**-Anweisung

- ▶ `(liste)`
- ▶ `{ liste;}`
- ▶ `funktionsname() { liste; }`

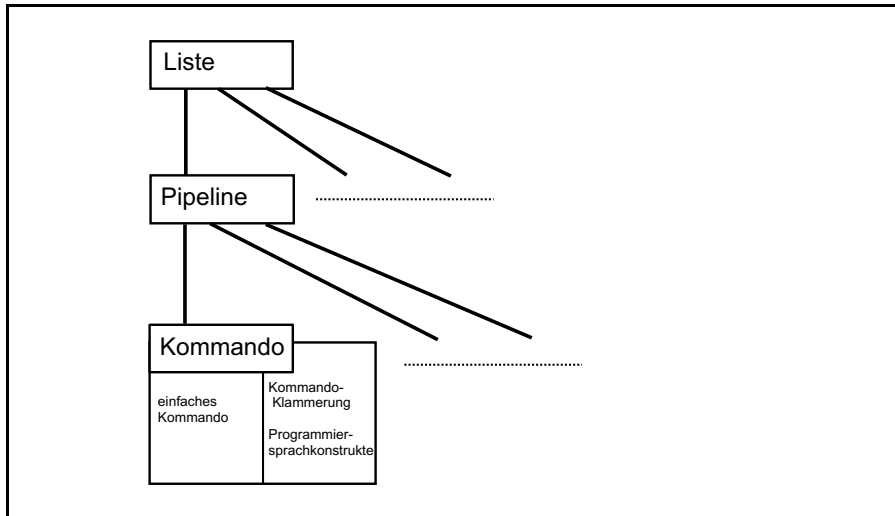


Abbildung 4.1: Listen, Pipelines und Kommandos

4.2.1 Einfache Kommandos und exit-Werte

Ein *einfaches Kommando* ist eine Folge von Wörtern, die durch Leer- und /oder Tabulatorzeichen voneinander getrennt sind. Das erste Wort gibt den Namen des auszuführenden Kommandos an. Bis auf wenige Ausnahmen (wie z.B. Wertzuweisungen an Variablen¹) werden die restlichen Wörter dem aufgerufenen Kommando als Argumente übergeben: *kommandoname argument1 argument2*

Der *kommandoname* wird dabei auch als *argument0* bezeichnet.

Beispiel

`ls` [Liste aller Dateien eines Directorys
(keine Angabe von Argumenten)]

`echo Hallo Emil` Ausgabe: Hallo Emil
 | |
 arg1 arg2

`echo "Hallo Emil," " wie geht es ?"` Ausgabe: Hallo Emil, wie geht es es ?
 | |
 arg1 arg2

1. Wird im Kapitel 4.6.4 ausführlich behandelt.

Der Rückgabewert eines Kommandos wird als sein *exit-Status* bezeichnet. Der *exit-Status* eines Kommandos zeigt immer den Erfolgsgrad der Kommandoausführung an:

Status	Anzeige
0	zeigt an, daß entsprechendes Kommando erfolgreich ausgeführt wurde ^a .
verschieden von 0	zeigt an, daß das Kommando nicht erfolgreich ausgeführt werden konnte. Der <i>exit-Status</i> liefert in diesem Fall noch zusätzlich die Information, ob das Kommando normal (z.B. Datei nicht vorhanden) oder abnormal (z.B. bei Division durch 0 oder bei Abbruch durch den Benutzer) beendet wurde. Wurde ein Programm abnormal beendet, so wird auf den eigentlichen <i>exit-Status</i> noch 0200 (oktal) ^b aufaddiert.

- a. Achtung: Dies ist ein wesentlicher Unterschied zur Programmiersprache C, bei der der Wert 0 als FALSE und alle anderen Werte als TRUE interpretiert werden; gerade diese Eigenheit bereitet manchen C-Programmierern anfänglich Schwierigkeiten beim Arbeiten mit der Shell.
- b. dezimal: 128

Mit dem Aufruf

echo \$?

kann der *exit-Status* des zuletzt ausgeführten Kommandos ausgegeben werden.

Beispiel

Für dieses Beispiel ist ein C-Programm *div_null.c*¹ zu erstellen, das eine Division durch 0 enthält:

```
#include <stdio.h>
main()
{
    int divi, a=5, b=0;

    divi=a/b;
    printf("Die Division von %d / %d = %d\n", a, b, divi);
}

$ pwd [↵] [Gib Working-Directory aus]
/user1/egon/shellueb [Ausgabe des Working-Directorys]
$ ls div_null.c [↵] [Liste Dateinamen div_null.c]
div_null.c [Ausgabe des Dateinamens div_null.c]
$ cc -o div_null div_null.c [↵] [Kompiliere div_null.c]
$ echo $? [↵] [Gib exit-Status der Kompilierung aus!]
0 [Ausgabe von 0 (Kompilierung erfolgreich)]
$ cat add.c [↵] [Gib Datei add.c aus]
```

1. Muß mit einem Editor, wie etwa **vi** oder **ed** erstellt werden.

```

cat: cannot open add.c      [ Meldung: add.c kann nicht eröffnet werden]
$ echo $?                  [Gib exit-Status des letzt. Kdos (cat) aus.]
2                          [ Ausgabe von 2 (cat war nicht erfolgreich)]
$ find / -name "*.c" -print [Suche alle C-Dateien des Dateisystems]
Strg-\                    [ Abbruch des find-Kommandos]
$ echo $?                  [Gib exit-Status des letzt. Kdos (find) aus.]
131                        [ Ausgabe von 131 (find abnormal beendet:
                          128+3)]
$ div_null                 [Aufruf von div_null]
Illegal instruction - core dumped [ Meldg., illegaler Bef. (Div. durch 0)]
$ echo $?                  [Gib exit-Status des letzt. Kdos (div_null)
                          aus.]
136                        [ Ausgabe von 132 (div_null abnormal beendet:
                          128+8)]
$

```

Hinweis

Da es viele Ursachen geben kann, warum ein Kommando nicht erfolgreich ablaufen konnte, verwenden viele UNIX-Kommandos den exit-Status, um den Grund für den Mißerfolg mitzuteilen¹. **grep** etwa liefert den exit-Status

- 0 wenn etwas gefunden wurde
- 1 wenn nichts gefunden wurde
- 2 wenn in der Kommandozeile ein Syntaxfehler vorlag, oder es auf die Datei, in der es suchen soll, nicht zugreifen kann.

```

$ grep main div_null.c
main()
$ echo $?
0
$ grep teile div_null.c
$ echo $?
1
$ grep main divnul.c
grep: can't open divnul.c
$ echo $?
2
$

```

4.2.2 Pipelines

Eine *Pipeline* ist eine Folge von einem oder mehreren Kommandos, welche mit | voneinander getrennt sind. Das Pipesymbol | bewirkt – wie bereits im ersten Buch besprochen –, daß die Standardausgabe des links vom Pipesymbol | angegebenen Kommandos direkt in die Standardeingabe des rechts davon stehenden Kommandos weitergeleitet wird.

1. Wozu man diesen exit-Status benötigt, wird bei der Vorstellung des Kommandos **test** in Kapitel 4.10 deutlich.

Beispiel `ls /bin | wc -w`

_____ Kommando
 _____ Pipesymbol
 _____ Kommando¹

Bei dieser Kommandozeile wird die Standardausgabe des **ls**-Kommandos direkt in die Standardeingabe des **wc**-Kommandos weitergeleitet:

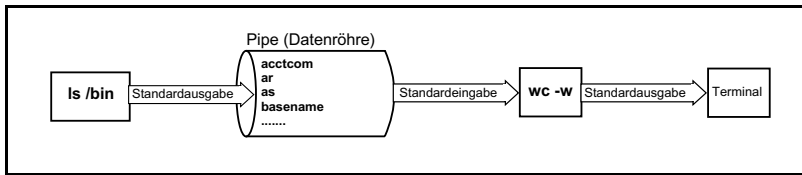


Abbildung 4.2: Pipeline für `ls /bin | wc -w`

Eine Pipeline ist allerdings nicht nur auf zwei Kommandos beschränkt, sondern kann sich über eine ganze Kommando-Kette erstrecken. Bei der Angabe der Kommandozeile

ls /bin | sort -r | pg²

würde z.B. die Standardausgabe von **ls** in die Standardeingabe von **sort** weitergeleitet. Das von **sort** auf die Standardausgabe geschriebene Ergebnis wird dann direkt in die Standardeingabe von **pg** weitergeleitet.

Eine Pipe wird vom System über einen Puffer realisiert. Das linke Kommando schreibt in den Puffer, während zu gleicher Zeit das rechte Kommando aus dem Puffer liest. Kommandos, die über eine Pipe verknüpft sind, laufen also parallel (als eigene Prozesse) ab, und die Shell wartet auf die Beendigung des letzten Kommandos, bevor sie mit der Abarbeitung der nächsten Kommandozeile fortfährt.

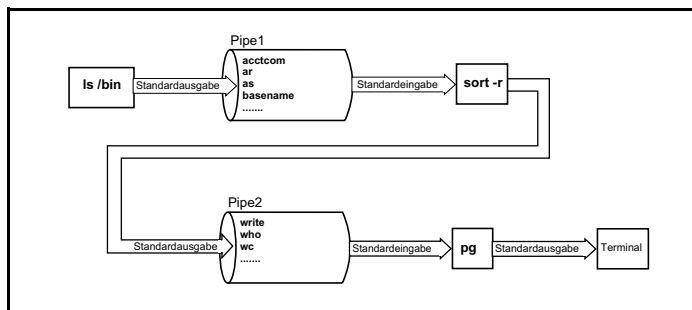


Abbildung 4.3: Pipeline für `ls /bin | sort -r | pg`

1. Erinnerung: Kommandos umfassen nicht nur einfache Kommandos.
2. Falls das Kommando **pg** auf Ihrem System nicht vorhanden ist, verwenden sie stattdessen das Kommando **more**

Der exit-Status einer Pipeline ist der exit-Status des zuletzt in der Pipeline angegebenen Kommandos.

Beispiel	<pre> \$ pwd /user1/egon/shellueb \$ cat add.c cat: cannot open add.c \$ echo \$? 2 \$ cat add.c grep hans cat: cannot open add.c \$ echo \$? 1 \$ cat div_null.c grep hans \$ echo \$? 1 \$ </pre>	<pre> [Gib Working-Directory aus] [Ausgabe des Working-Directorys] [Gib Datei add.c aus] [Meldung: add.c kann nicht eröffnet werden] [Gib exit-Status des letzt. Kdos (cat) aus] [Ausgabe von 2 (cat war nicht erfolgreich)] [Suche in Datei add.c den String "hans"] [Meldung: add.c kann nicht eröffnet werden] [Gib exit-Status des letzt. Kdos (Pipe) aus] [Ausgabe von 1 ("hans" nicht gefunden)] [Suche in Datei div_null.c den String "hans"] [Gib exit-Status des letzt. Kdos (Pipe) aus] [Ausgabe von 1 ("hans" nicht gefunden)] </pre>

4.2.3 Listen

Eine *Liste* ist eine Folge von ein oder mehreren Pipelines, welche durch die Zeichen

;, &, && oder ||

voneinander getrennt sind. & und ; dürfen dabei auch am Ende einer solchen Liste angegeben sein.

Diese Trennzeichen haben dabei folgende Bedeutung:

Das Zeichen ; bewirkt, daß die in der Liste angegebenen Pipelines streng nacheinander ausgeführt werden. Anstelle von ; könnte auch `\n` (Neuezeile-Zeichen) und dann das nächste Kommando eingegeben werden. Der einzige Unterschied zu dieser typischen Eingabe (ein Kommando eingeben, auf Ausführungsende warten, dann nächstes Kommando eingeben, bis zum Ausführungsende warten, usw.) ist, daß man bei ; mehrere Kommandos in einer Kommandozeile eingeben kann, bevor mit der sequentiellen Ausführung der angegebenen Kommandos begonnen wird.

Beispiel	<pre> \$ ls; echo "Dateianzahl: \c";ls wc -w core div_null div_null.c Dateianzahl: 3 \$ ls core div_null div_null.c \$ echo "Dateianzahl: \c" Dateianzahl: \$ ls wc -w 3 \$ </pre>	<pre> [Linux: ls; echo -e "..."; ..] [In Linux (bash): echo -e "Dateianzahl: \c"] </pre>

Das Zeichen **&** bewirkt, daß die davor angegebene Pipeline¹ im Hintergrund (asynchron) ausgeführt wird. Das heißt, daß die Shell nicht auf das Ende der in der Pipeline angegebenen Kommandos wartet, sondern sofort mit der Abarbeitung (bzw. Entgegennahme) des nächsten Kommandos fortfährt, während die vorherige Pipeline parallel dazu im Hintergrund abläuft.

Beispiel

```
$ cc -o div_null div_null.c & lp div_null.c ↵
1257                                [ <-- PID für Hintergrund-Prozeß]
request id is <druckername> (1 file)
$ ls|wc & echo "Hallo, ich bin fast fertig" ↵
1286                                [ <-- PID für Hintergrund-Prozeß]
Hallo, ich bin fast fertig
3                                  [ <-- Anzahl der Dateien]
$
```

Das Zeichen **&&** bewirkt, daß die danach angegebenen Pipelines nur dann ausgeführt werden, wenn die vorherigen Pipelines erfolgreich (exit-Status = 0) ausgeführt werden konnten.

Beispiel

```
$ cc -o div_null div_null.c && div_null ↵
Illegal instruction - core dumped
$ cc -o div_null divnull.c && div_null ↵
0: No source file divnull.c
$ ls *.c && cp *.c /tmp && echo "C-Quellen ->/tmp kopiert" ↵
div_null.c
C-Quellen ->/tmp kopiert
$ ls *.h && cp *.h /tmp && echo "C-Header ->/tmp kopiert" ↵
*.h: No such file or directory
$
```

Das Zeichen **||** bewirkt, daß die danach angegebenen Pipelines nur dann ausgeführt werden, wenn die vorherigen Pipelines nicht erfolgreich (exit-Status verschieden von 0) ausgeführt werden konnten.

Beispiel

```
$ grep division div_null.c || grep divi div_null.c ↵
int divi, a=5, b=0;
divi=a/b;
printf("Die Division von %d / %d = %d\n", a, b, divi);
$ cc -o div_null div_null.c && div_null || vi div_null.c ↵
Illegal instruction - core dumped [Aufruf vi div_null.c; mit ZZ verlassen]
$
```

Die Vorrangregeln der hier angegebenen Trennzeichen von Pipelines in einer Liste sind:

(; gleich &) < (&& gleich ||)

1. Eine Pipeline kann sich auch aus nur einem Kommando zusammensetzen.

4.2.4 Syntaxdiagramme zur Pipeline und Liste

Die zuvor vorgestellten Begriffe werden nochmals in Form von vereinfachten Syntaxdiagrammen zusammengefaßt in Abbildung 4.4.

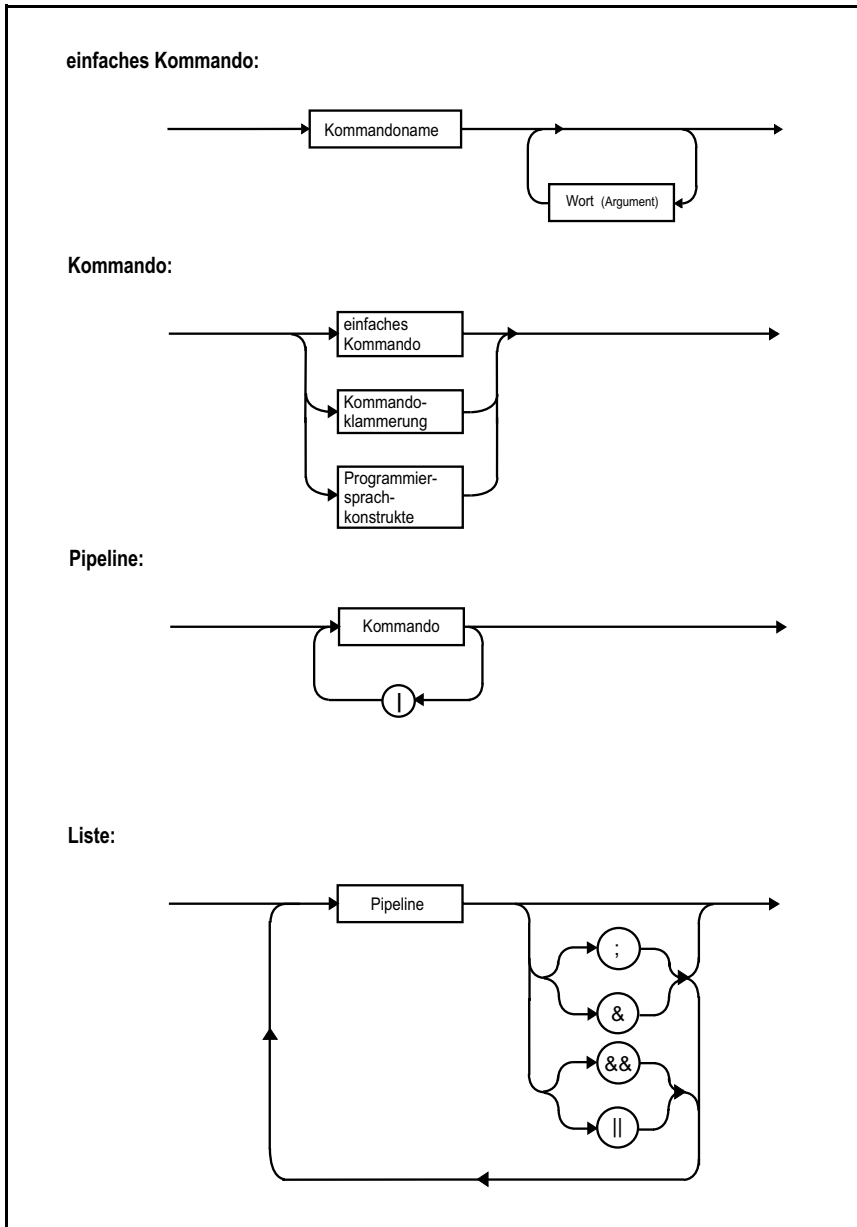


Abbildung 4.4: Syntaxdiagramme zur Pipeline und Liste

Hinweis

Die hier gegebenen Syntaxdiagramme beziehen sich auf eine vollständige Kommandozeile. Eine Kommandozeile kann sich dabei auch über mehrere Eingabezeilen erstrecken, z.B. wenn das Fortsetzungszeichen \ als letztes Zeichen einer Zeile angegeben wurde.

Wird eines der Symbole |, && oder || an letzter Stelle einer Zeile angegeben, so nimmt die Shell in diesem Fall an, daß die Kommandozeile noch nicht vollständig ist und fordert den Benutzer durch Ausgabe des Sekundärpromptzeichens > zur weiteren Eingabe auf.

Beispiel

```
$ ls | 
> wc -w 
4
$
```

4.3 Kommentare

Beginnt ein Wort mit #, so werden dieses Wort und alle nachfolgenden Zeichen dieser Zeile von der Shell als Kommentar interpretiert und als solcher von ihr ignoriert.

Die Angabe von Kommentaren macht allerdings bei einer interaktiven Eingabe wenig Sinn. Erst bei den im nächsten Kapitel vorgestellten Shell-Skripts wird das Vorhandensein eines Kommentarzeichens gerechtfertigt.

Beispiel

```
$ # Ich rufe jetzt das 
$ # ls-Kommando auf 
$ ls # ls-Aufruf 
core
div_null
div_null.c
$ echo Das Nummernzeichen# ist Kommentar 
Das Nummernzeichen# ist Kommentar [# ist nicht Anfang eines Worts]
$ echo Das Nummernzeichen # ist Kommentar 
Das Nummernzeichen
$ echo Das Nummernzeichen \# ist Kommentar 
Das Nummernzeichen # ist Kommentar [Sonderbedeutung von # wurde ausgeschaltet]
$
```


4.4 Shell-Skripts (Shell-Prozeduren)

In UNIX können Kommandos (bzw. Pipelines oder Listen) nicht nur interaktiv eingegeben werden, sondern es ist auch möglich, sie in einer Datei abzulegen und diese Datei dann der Shell zur Abarbeitung der darin angegebenen Kommandos vorzulegen.

Solche Kommandodateien, in anderen Betriebssystemen als Batch-Dateien oder Submit-Dateien bezeichnet, werden unter UNIX *Shell-Skripts* (oder manchmal auch Shell-Prozeduren) genannt.

4.4.1 Aufruf von Shell-Skripts

UNIX bietet zwei Möglichkeiten, Shell-Skripts zu starten und somit die darin enthaltenen Kommandos zur Ausführung zu bringen:

Aufruf mit sh

`sh skript-name`

Beispiel

```
$ cat gruss ↵1
echo "Guten Tag, ich bin ein einfaches Shell-Skript"
$ sh gruss ↵
Guten Tag, ich bin ein einfaches Shell-Skript
$ cat lszaehl ↵
ls
echo "Dateianzahl: \c"    # unter Linux (bash): echo -e "Dateianzahl: \c"
ls | wc -l
$ sh lszaehl ↵
core
div_null
div_null.c
gruss
lszaehl
Dateianzahl:      5
$
```

Wird ein Shell-Skript mit **sh** aufgerufen, so wird eine neue Subshell² gestartet, die das angegebene Shell-Skript ausführt.

-
1. Die Datei *gruss* ist zuerst mit einem Editor (z.B. **vi** oder **ed**) zu erstellen. Auf den folgenden Seiten werden die entsprechenden Shell-Skripts nur mit **cat** ausgegeben, und es wird nicht mehr explizit darauf hingewiesen, daß sie zuvor mit Hilfe eines Editors erstellt bzw. von der in Kapitel 1.2 angegebenen *www*-Seite heruntergeladen werden müssen.
 2. Kindprozeß zur aktuellen Shell.

Beispiel `$ cat zeig_pid`
echo "Ich habe die PID: \c" # unter Linux (bash): echo -e "...\\c"
echo \$\$
`$ echo $$`
192 [Ausgabe der PID für gerade aktive Shell]
`$ sh zeig_pid`
Ich habe die PID: 215 [Ausgabe der PID für Shell-Skript (Subshell)]
`$ echo $$`
192 [Ausgabe der PID für gerade aktive Shell]
\$

Alleinige Angabe des Shell-Skript-Namens (ohne sh)

skript-name

Bei dieser Aufrufform muß allerdings die Datei, in der das Shell-Skript gespeichert ist, ausführbar sein, was beim Aufruf mit **sh** nicht erforderlich ist.

Falls die entsprechende Shell-Skript-Datei nicht ausführbar ist, so muß sie zuerst mit dem Kommando **chmod** dazu gemacht werden.

Beispiel `$ gruss`
gruss: execute permission denied
`$ chmod u+x gruss`
`$ gruss`
Guten Tag, ich bin ein einfaches Shell-Skript
\$

Wie beim Aufruf mit **sh** wird auch hier eine eigene Subshell gestartet, welche die Ausführung des aufgerufenen Shell-Skripts übernimmt.

Beispiel `$ echo $$`
192 [Ausgabe der PID für gerade aktive Shell]
`$ zeig_pid`
zeig_pid: execute permission denied
`$ chmod u+x zeig_pid`
`$ zeig_pid`
Ich habe die PID: 273 [Ausgabe der PID für Shell-Skript (Subshell)]
`$ echo $$`
192 [Ausgabe der PID für gerade aktive Shell]
\$

4.4.2 Namen für Shell-Skripts

Als Dateinamen für Shell-Skripts sollten keine Namen von UNIX-Kommandos gewählt werden, da sonst eventuell¹ das Shell-Skript und nicht das entsprechende UNIX-Kommando aufgerufen wird.

1. Abhängig vom Inhalt der **PATH**-Variablen (siehe Kapitel 4.6.2) und dem Kommandotyp (siehe Kapitel 4.15, wo die builtin-Kommandos vorgestellt werden).

Beispiel `$ cat wc`
echo "Die Toilette befindet sich in der 2.Etage rechts vom Aufzug"
`$ chmod u+x wc`
`$ wc -l div_null.c`
Die Toilette befindet sich in der 2.Etage rechts vom Aufzug
`$ rm wc`
\$

Eine andere Gefahr bei der Wahl eines Kommandonamens als Skriptname ist, daß sich das aufgerufene Shell-Skript ständig selbst wieder aufruft und sich somit in eine »Endlos-Rekursion« begibt, die nach einiger Zeit mit der Fehlermeldung

fork failed - too many processes

beendet wird.

Beispiel `$ cat ls`
echo "Directory:\c " # unter Linux (bash): echo -e "Directory:\c"
pwd
ls -CF
echo "-----"
`$ chmod u+x ls`
`$ ls`
Directory: /user1/egon/shellueb
Directory: /user1/egon/shellueb
Directory: /user1/egon/shellueb
:
:
:
Directory: /user1/egon/shellueb
Directory: /user1/egon/shellueb
Directory: /user1/egon/shellueb
ls: fork failed - too many processes
\$

Der in der 3. Zeile des Skripts angegebene **ls**-Aufruf führt zu einem rekursiven Aufruf des Skripts **ls** und nicht zum Aufruf des UNIX-Kommandos **ls**. Da dieses erneute Aufrufen immer eine neue Subshell startet, sprich einen neuen Kindprozeß (mit **fork()**) kreiert, wird schließlich irgendwann die Systemgrenze von maximal möglichen Prozessen erreicht, die parallel ablaufen können. Dies führt dann zum Programmabbruch mit der obigen Fehlermeldung (siehe Abbildung 4.5).

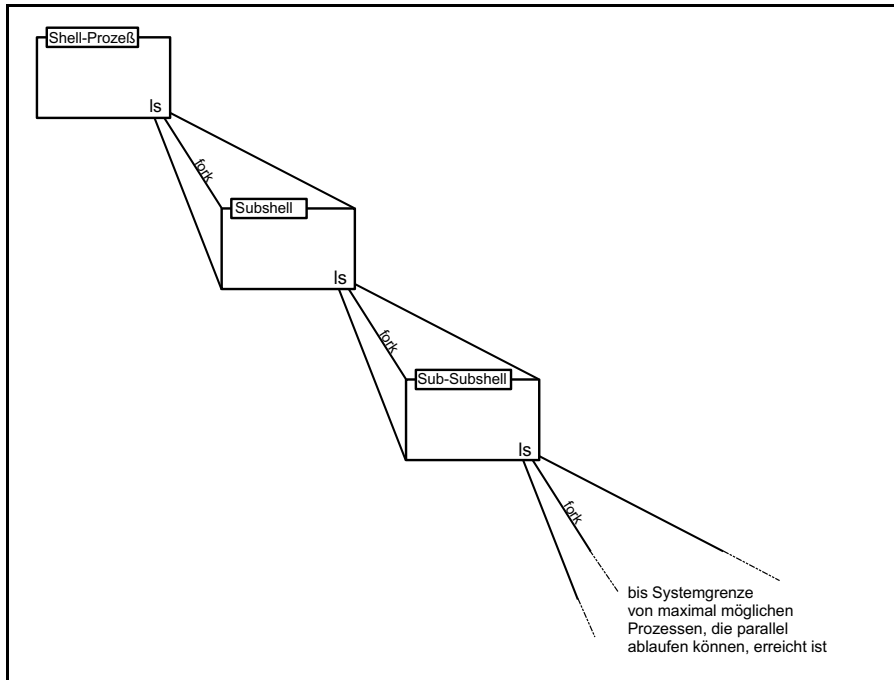


Abbildung 4.5: Endlos-Rekursion in einem Shell-Skript

Dieses Problem könnte durch Angabe des absoluten Pfadnamens für das entsprechende UNIX-Kommando behoben werden:

```
$ cat ls
echo "Directory:\c "      # unter Linux (bash): echo -e "Directory:\c"
pwd
/bin/ls -CF              # Aufruf des UNIX-Kdos ls (Pfadname /bin/ls)
echo "-----"
$ ls
core          div_null.c  ls*           zeig_pid*
div_null*     gruss*        lszaehl
-----
$ rm ls        [Shell-Skript ls wieder löschen]
$
```

4.4.3 Hinweise zu Shell-Skripts

Neben den beiden Aufrufformen, die zuvor angegeben wurden, könnte ein Shell-Skript auch mit

```
sh <skript-name
```

aufgerufen werden. Diese Aufrufform unterscheidet sich allerdings in einem wichtigen Punkt von den beiden anderen Aufrufmöglichkeiten: Für die Dauer der Skript-Ausführung ist die Standardeingabe nicht mehr auf das Terminal eingestellt, sondern in die Skript-Datei umgelenkt.

Beispiel

```
$ cat eing_zaehl ↵  
cat | wc  
$
```

Beim Aufruf mit

```
sh eing_zaehl
```

würde **cat** von der Standardeingabe, die auf die Tastatur voreingestellt ist, lesen, bis ein EOF (*Strg-D*) eingegeben wird. Der eingegebene Text wird dann über eine Pipe an das Kommando **wc** weitergeleitet:

```
$ sh eing_zaehl ↵  
Das ist ein ↵  
einfacher Text ↵  
[Strg-D]  
      2      5      27  
$
```

Dagegen würde beim Aufruf

```
sh <eing_zaehl
```

cat von der nach *eing_zaehl* umgelenkten Standardeingabe lesen. Da *eing_zaehl* bei diesem Aufruf von der Shell als Kommandodatei interpretiert wird, ist es ihr nicht möglich, diese gleichzeitig als Daten-Datei zu interpretieren. Jedenfalls wäre bei dieser Aufrufform keine Eingabe eines Textes über Tastatur möglich:

```
$ sh <eing_zaehl ↵  
      0      0      0  
$
```

Diese Problematik wird in Kapitel 4.9 (Ein- und Ausgabeumlenkung) ausführlicher behandelt.

Shell-Skripts sollten (wie auch Programme in höheren Programmiersprachen) immer kommentiert sein. Dazu steht das zuvor vorgestellte Kommentarzeichen **#** zur Verfügung.

```

Beispiel $ cat ll
# --- Auflisten der Dateien im Working-Directory
# --- mit den Optionen -CF; zusätzlich wird noch
# --- die Anzahl der ausgegebenen Dateinamen gemeldet
#
# Erstellt von: Helmut Herold
# am: 11.11.1996
ls -CF
echo "---- Dateianzahl: \c" # unter Linux (bash): echo -e "... \c"
ls | wc -w
$ chmod u+x ll
$ ll
core          div_null.c          gruss*          lszaehl
div_null*     eing_zaehl          ll*             zeig_pid*
---- Dateianzahl: 8
$

```

4.5 Kommandosubstitution

4.5.1 Definition

In manchen Anwendungsfällen kann es nützlich sein, die Ausgabe eines Kommandos als Teil der Kommandozeile interpretieren zu lassen. Dazu bietet die Shell die sogenannte Kommandosubstitution an:

Kommandos, deren Standardausgabe von der Shell als Teil der Kommandozeile zu verwenden ist, müssen mit »Gegen-Apostrophen«¹:

``kommando``

geklammert werden.

```

Beispiel $ echo Heute ist der `date '+%d.%m.%y (%a)'\`
Heute ist der 5.11.96 (Tue)
$

```

Erklärung:

echo Heute ist der `date '+%d.%m.%y (%a)'\`

wird zuerst ausgeführt und liefert die Ausgabe:

05.11.96 (Tue)

die dann – aufgrund der Kommandosubstitution ``...`` – im echo-Kommando eingesetzt wird, so daß schließlich folgender Befehl daraus resultiert:

echo Heute ist der 5.11.96 (Tue)

1. Engl.: *backquotes* oder *accents graves*

Dieser Befehl führt dann zur obigen Ausgabe.

```
$ echo Zur Zeit arbeiten `who | wc -l` Benutzer am System
Zur Zeit arbeiten 5 Benutzer am System
$
```

Erklärung:

```
echo Zur Zeit arbeiten `who | wc -l` Benutzer am System
```

wird zuerst ausgeführt und bewirkt, daß die Ausgabe von **who** über eine Pipe an das Kommando **wc -l** weitergeleitet wird, so daß alle Zeilen (Benutzer) des **who**-Kommandos gezählt und ausgegeben würden:

```
5
```

Diese Ausgabe wird dann – aufgrund der Kommandosubstitution ``...`` – im **echo**-Kommando eingesetzt wird, so daß schließlich folgender Befehl daraus resultiert:

```
echo Zur Zeit arbeiten 5 Benutzer am System
```

Dieser Befehl führt dann zur obigen Ausgabe.

4.5.2 Unterschied zwischen Pipeline und Kommandosubstitution

Da der Unterschied zwischen einer Pipeline und der Kommandosubstitution manchen UNIX-Anwendern anfänglich Schwierigkeiten bereitet, soll das nachfolgende Beispiel helfen, diesen Unterschied zu verdeutlichen: Eine Datei *zaehle.txt* soll Namen von Dateien enthalten, zu denen die darin enthaltene Wortzahl zu bestimmen ist:

```
$ cat zaehle.txt
/etc/magic
/etc/inittab
/usr/include/stdio.h
$
```

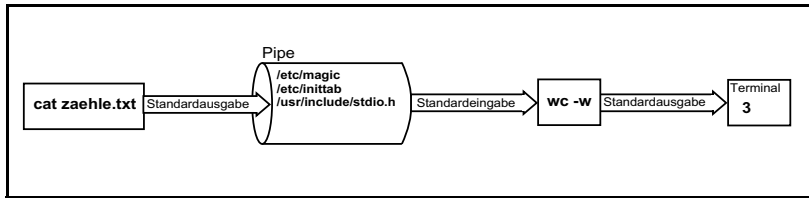
Der Aufruf

```
cat zaehle.txt | wc -w
```

liefert dann die falsche Ausgabe

```
3
```

da in diesem Fall nicht der Inhalt der in *zaehle.txt* genannten Dateien, sondern der Inhalt von *zaehle.txt* selbst ausgewertet wird (siehe Abbildung 4.6):

Abbildung 4.6: Pipeline zu `cat zaehle.txt | wc -w`

Mit dem Aufruf

```
wc -w `cat zaehle.txt`
```

dagegen wird – wie gefordert – der Inhalt der in `zaehle.txt` angegebenen Dateien und nicht der Inhalt von `zaehle.txt` selbst ausgewertet. Nach der Durchführung der Kommandosubstitution `cat zaehle.txt`` würde folgende Kommandozeile aus der obigen resultieren:

```
wc -w /etc/magic /etc/inittab /usr/include/stdio.h
```

und z.B. folgende Ausgabe liefern:

```
572 /etc/magic
112 /etc/inittab
365 /usr/include/stdio.h
1049 total
```

4.5.3 Metazeichen in der Kommandosubstitution

Alle Metazeichen behalten innerhalb einer Kommandosubstitution ihre Sonderbedeutung.

Beispiel

```
wc -w `cat zaehle.txt | grep "/etc"`
```

würde nur den Inhalt der in `zaehle.txt` angegebenen Dateien zählen, deren Name den String `"/etc"` enthält:

```
572 /etc/magic
112 /etc/inittab
684 total
```

Die Sonderbedeutung des Metazeichens ``` kann durch Voranstellen von `\` ausgeschaltet werden.

Beispiel

```
$ echo Mein Home-Directory ist: `~/user1/egon`  
Mein Home-Directory ist `~/user1/egon`  
$
```


4.5.4 Schachtelung von Kommandosubstitutionen

Eine Schachtelung von Kommandosubstitutionen ist mit der Angabe von `\`...\`` möglich. Da bei der ersten Auswertung der Kommandozeile alle vorangestellten `\`` von der Shell entfernt werden, wird hierdurch eine erneute (geschachtelte) Kommandosubstitution `\`...\`` aufgedeckt, die vor der umschließenden Kommandosubstitution ausgewertet wird.

Beispiel

Das Shell-Skript *cph* kopiert die erste System-Header-Datei, die im C-Programm *div_null.c* verwendet wird, in das Working-Directory:

```
$ cat cph
cp `find /usr/include -name \
  \`grep "#include" div_null.c | line | cut -f2- -d" " | \
  tr -d "<>" \` \
  -print` .
$ chmod u+x cph
$ cph
$ ls stdio.h
stdio.h
$
```

Erklärung:

```
cp `find /usr/include -name \
  \`grep "#include" div_null.c | line | cut -f2- -d" " | \
  tr -d "<>" \` \
  -print` .
                                     |
                                     | 1. Kommandosubstitution
                                     |
cp `find /usr/include -name stdio.h -print` .
                                     |
                                     | 2. Kommandosubstitution
                                     |
cp /usr/include/stdio.h .
```

Hinweis

Die Wirkung der Kommandosubstitution ``cat datei`` kann auch durch die Angabe `< datei` erreicht werden, wobei die zweite Form eine wesentlich schnellere Variante darstellt.

```
wc -w < zaehle.txt | grep "/etc"
572 /etc/magic
112 /etc/inittab
684 total
```

4.6 Shell-Parameter

Die Shell kennt zwei Arten von Parametern:

Positionsparameter

Ihr Name wird als Ziffer 0, 1, 2, ..., 9 angegeben.

Shell-Variablen (oder Schlüsselwort-Parameter)

Ihr Name ist

- ▶ ein Bezeichner (siehe Kapitel 3) oder
- ▶ eines der Zeichen * @ # - ? \$!

Durch Voranstellen von \$ vor einem Parameternamen wird der Wert angesprochen, der unter diesen Parameternamen gespeichert ist:

\$parameter entspricht: Wert von *parameter*

Alle Werte werden dabei als Strings gespeichert; so wird z.B. 93 in zwei Bytes mit den Zeichen '9' und '3' gespeichert und nicht als numerisches Bitmuster für 93. Mittels spezieller Funktionen (wie z.B. **expr**¹) kann ein solcher String-Wert jedoch zum Zwecke arithmetischer Berechnungen auch als numerischer Wert interpretiert werden.

4.6.1 Positionsparameter

Positionsparameter stellen die an ein Shell-Skript übergebenen Argumente zur Verfügung, wobei das 1. Argument dem Parameter 1, das 2. Argument dem Parameter 2, usw. zugewiesen wird. Dem Parameter 0 wird der Name des aufgerufenen Shell-Skripts zugewiesen. Auf die Werte der einzelnen Parameternamen kann – wie schon erwähnt – durch Voranstellen des \$-Zeichens zugegriffen werden.

Beispiel

```
$ cat ausgab ↵
echo Das erste Argument ist $1
echo Das zweite Argument ist $2
echo Das Skriptname ist $0
$ chmod u+x ausgab ↵
$ ausgab hans fritz franz ↵
Das erste Argument ist hans
Das zweite Argument ist fritz
Das Skriptname ist ausgab
$
```

1. Siehe Kapitel 4.10.

Der Inhalt von Parameter 3 (**\$3**), der bei diesem Aufruf "franz" ist, wird vom Shell-Skript *ausgab* nicht benutzt.

Der Aufruf

```
ausgab "Hans Meyer" "Fritz Golke"
```

würde folgende Ausgabe liefern:

```
Das erste Argument ist Hans Meyer
Das zweite Argument ist Fritz Golke
Das Skriptname ist ausgab
```

Explizites Setzen von Positionsparametern

Mit dem builtin-Kommando **set**¹ können den Positionsparametern auch explizit Werte zugewiesen werden. Die beim Aufruf von **set** angegebenen Argumente werden dabei in der Reihenfolge ihrer Angabe den Positionsparametern zugewiesen.

Der Positionsparameter 0 wird durch den **set**-Aufruf nicht neu gesetzt, sondern behält weiterhin als Wert den Namen der Shell »-sh«² bzw. des aufgerufenen Shell-Skripts (Dies läßt sich dadurch erklären, daß **set** kein eigenes Kommando, sondern ein builtin-Kommando (Programmteil) der Shell ist):

```
set argument1 argument2 argument3 ....
    |           |           |           |
    0           1           2           3   Positionsparameter
    |
    "-sh" bzw.
    Skriptname
```

Beispiel

```
$ echo $0 ↵
-sh
$ set Hund Katze Maus jagt die ↵
$ echo $0 ↵
-sh
$ echo $1 $4 $5 $2 ↵
Hund jagt die Katze
$ echo $2 $4 $5 $3 ↵
Katze jagt die Maus
$ cat datum ↵
set `date +%d %h %y`
echo Du hast $0 gestartet
echo Heute ist der $1.$2.$3
```

1. Builtin-Kommandos sind Bestandteil des Shell-Programms. Das heißt, daß diese »Kommandos« von der Shell selbst ausgeführt werden, und dafür keine eigene Subshell gestartet wird. Alle builtin-Kommandos der Shell werden im Kapitel 4.15.8 vorgestellt.
2. Das vorangestellte Minuszeichen - zeigt an, daß beim Start dieser Shell die Datei *.profile* im Home-Directory gelesen wurde. Dies gilt fast immer für eine Login-Shell.

```
$ chmod u+x datum ↵
$ datum ↵
Du hast datum gestartet
Heute ist der 08.Nov.96
$
```

Werden bei einem Skript- oder **set**-Aufruf mehr als 9 Argumente angegeben, so sind die restlichen zwar noch vorhanden, aber auf diese Argumente (10. Argument, 11. Argument, usw.) kann zunächst nicht zugegriffen werden.

```
$ cat countdown ↵
echo $9
echo $8
echo $7
echo $6
echo $5
echo $4
echo $3
echo $2
echo $1
$ chmod u+x countdown ↵
$ countdown one two three four five six seven eight nine ten eleven ↵
nine
eight
seven
six
five
four
three
two
one
$
```

Etwas später wird gezeigt, wie trotzdem (unter Verwendung des Kommandos **shift**) ein Zugriff auf diese weiteren Argumente möglich wird.

Werden innerhalb eines Shell-Skripts den Positionsparametern mit **set** neue Werte zugewiesen, so werden deren alte Inhalte (wie z.B. die Argumente aus der Kommandozeile) überschrieben.

Beispiel

Das nachfolgende Shell-Skript *anfang* gibt die ersten 7 Wörter einer Datei mit Bindestrich getrennt in einer Zeile aus:

```
$ cat anfang ↵
set `cat $1`
echo $1 - $2 - $3 - $4 - $5 - $6 - $7
$ chmod u+x anfang ↵
$ anfang div_null.c ↵
#include <stdio.h> - main() - { - int - divi, - a=5,
$
```

Bei den zu **set** angegebenen Argumenten findet Dateinamenexpandierung (siehe Kapitel 4.7) statt.

Beispiel

```
$ ls d*  
datum  
div_null  
div_null.c  
$ set d*  
$ echo $0  
-sh  
$ echo $1  
datum  
$ echo $2  
div_null  
$ echo $3  
div_null.c  
$ echo $4  
  
$
```

Werte der Positionsparameter verschieben

Mit dem Kommando **shift** können die Werte der Positionsparameter (nach links) verschoben werden:

shift [n]

Das Kommando **shift** bewirkt, daß die Werte der Positionsparameter um *n* Positionen nach vorne (links) geschoben werden. Wird *n* nicht angegeben, so wird 1 als Wert für *n* genommen. So bewirkt also ein **shift** ohne Argumente, daß der Wert **\$2** dem Positionsparameter **1**, der Wert **\$3** dem Positionsparameter **2**, usw. zugewiesen wird. Ein **shift 4** bewirkt dann, daß **\$5** dem Positionsparameter **1**, **\$6** dem Positionsparameter **2** usw. zugewiesen wird.

Beispiel

```
$ cat schiebe  
echo "$2 * \c" # unter Linux (bash): echo -e "$2 * \c"  
shift  
echo "$2 = \c" # unter Linux (bash): echo -e "$2 * \c"  
shift 3  
echo $2  
$ chmod u+x schiebe  
$ schiebe eins zwei drei vier fuenf sechs sieben acht  
zwei * drei = sechs  
$
```

Werden bei einem Skript- oder **set**-Aufruf mehr als 9 Argumente angegeben, so sind – wie zuvor erwähnt – die restlichen zwar noch vorhanden, aber auf diese Argumente (10. Argument, 11. Argument usw.) kann zunächst nicht zugegriffen werden. Unter Verwendung von **shift** ist nun ein Zugriff auf diese »überhängenden« Argumente möglich:

```
$ cat countdown2 ↵
eins=$1
zwei=$2
shift 2
echo "$9\n$8\n$7\n$6\n$5\n$4\n$3\n$2\n$1"  # Unter Linux: echo -e "..."
echo $zwei
echo $eins
echo $0
$ chmod u+x countdown2 ↵
$ countdown2 one two three four five six seven eight nine ten eleven ↵
eleven
ten
nine
eight
seven
six
five
four
three
two
one
countdown2
$
```

4.6.2 Shell-Variablen (Schlüsselwort-Parameter)

Der Name einer Shell-Variablen ist entweder als Bezeichner (siehe Abbildung 4.7) oder als eines der Zeichen

*** @ # - ? \$!**

anzugeben.

Um einer Shell-Variablen einen Wert zuweisen zu können, muß folgende Syntax verwendet werden:

*variablenname=wert*¹ [*variablenname=wert*]....

1. Vor und hinter dem = darf dabei kein Trennzeichen angegeben werden.

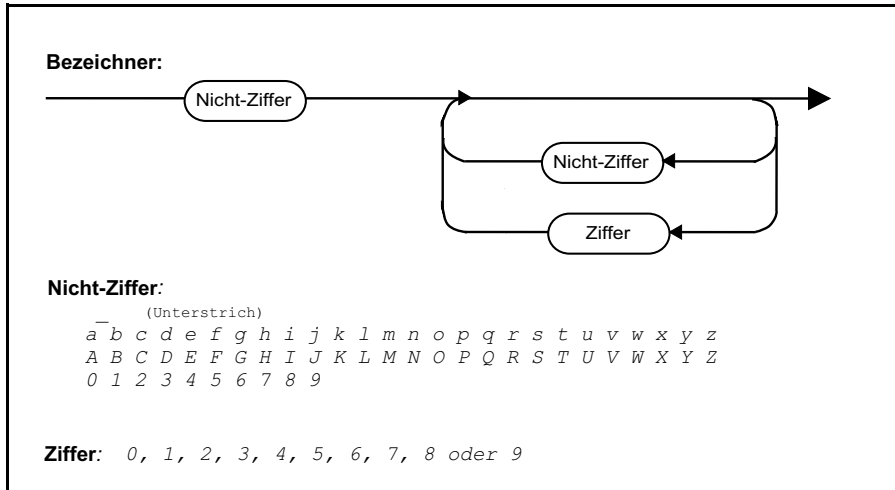


Abbildung 4.7: Syntax für einen Bezeichner

Um auf den Wert einer Shell-Variable zuzugreifen, ist wieder das Zeichen **\$** dem Variablennamen voranzustellen:

\$variablenname

Nachfolgend wird unterschieden zwischen

- ▶ vom Benutzer frei wählbaren und
- ▶ von der Shell vordefinierten Variablennamen.

Frei wählbare Variablennamen

Bezüglich der Wahl von benutzerdefinierten Variablennamen gelten die Regeln für Bezeichner (siehe Abbildung 4.7).

Beispiel

```

$ tier1=Hund
$ tier2=Katze
$ tier3=Maus
$ verb=jagt
$ artikel=die
$ echo $tier1 $verb $artikel $tier2
Hund jagt die Katze
$ echo $tier2 $verb $artikel $tier3
Katze jagt die Maus
$ wdir=`pwd`
$ echo $wdir
/user1/egon/shellueb
$ cd /bin
$ pwd
/bin

```

```
$ cd $wdir ↵
$ pwd ↵
/user1/egon/shellueb
$
```

In Variablen können auch ganze Kommandozeilen gespeichert werden, welche dann unter Angabe

\$variablenname

aufgerufen werden können.

Beispiel

```
$ mcc="cc -o div_null2 div_null.c" ↵
$ $mcc ↵
$ div_null2 ↵
Illegal instruction - core dumped
$
```

Wird auf den Wert einer Variablen zugegriffen, bevor ihr explizit ein Wert zugewiesen wurde, dann liefert ein solcher Zugriff die leere Zeichenkette:

```
$ echo $tier5 ↵
$
```

Soll einer Shell-Variablen eine Zeichenkette zugewiesen werden, in der die Bedeutung aller Sonderzeichen wie \$, Leerzeichen, usw. auszuschalten ist, so ist diese Zeichenkette mit '...' zu klammern.

Beispiel

```
$ hund=dackel ↵
$ tier1=$hund ↵
$ echo $tier1 ↵
dackel
$ tier1='$hund' ↵
$ echo $tier1 ↵
$hund
$ satz=Hund jagt die Katze ↵
jagt: not found
$ echo $satz ↵

$ satz='Hund jagt die Katze' ↵
$ echo $satz ↵
Hund jagt die Katze
$
```

Bei der Zuweisung eines Wertes an eine Shell-Variable findet – anders als beim **set**-Aufruf – keine Dateinamenexpansion statt:

```
$ c_dateien=c* ↵
$ echo "$c_dateien" ↵
c*
$
```


Wird allerdings der Wert einer Variablen, die Expanderungszeichen enthält, einem Kommandoaufruf als Argument (ohne Ausschalten der Sonderzeichen durch "..."-Klammerung) übergeben, so findet dort eine Dateinamenexpansion statt.

```
$ echo $c_dateien ↵
core countdow2 countdown cph
$ cd /bin ↵
$ echo $c_dateien ↵
cat cc chgrp chmod chown cmp conv convert cp cpio cprs csh
$ cd ↵
$ cd shellueb ↵
$
```

Ist eine Dateinamenexpansion bereits bei der Zuweisung erwünscht, so ist Kommandosubstitution zu verwenden:

```
$ c_dateien=`echo c*` ↵
$ echo "$c_dateien" ↵
core countdow2 countdown cph
$ echo $c_dateien ↵
core countdow2 countdown cph
$ cd /bin ↵
$ echo $c_dateien ↵
core countdow2 countdown cph
$ cd ↵
$ cd shellueb ↵
$
```

Die Definition einer Shell-Variablen kann mit dem builtin-Kommando **unset** wieder aufgehoben werden:

```
$ tier1="Elefant" ↵
$ echo $tier1 ↵
Elefant
$ unset tier1 ↵
$ echo $tier1 ↵

$
```

Das Entfernen von nicht mehr benötigten Shell-Variablen ist besonders bei umfangreichen Shell-Skripts sinnvoll, um nicht mehr benutzten Speicherplatz wieder freizugeben.

Lesen von der Standardeingabe

Das Kommando

read [*variable(n)*]

liest eine Zeile¹ von der Standardeingabe. Das erste Wort der Eingabezeile wird dann der ersten angegebenen *variable*, das zweite der zweiten *variable* usw. zugewiesen. Wenn mehr Worte als *variable(n)* angegeben sind, dann werden alle restlichen Worte der zuletzt angegebenen *variable* zugewiesen. Zur Ermittlung von Wortgrenzen werden die Zeichen aus der Shell-Variable **IFS**² verwendet.

Wenn **read** ohne Argumente angegeben wird, dann liest es zwar die ganze Eingabezeile, speichert den gelesenen Text aber nirgends ab.

read liefert nur dann einen von 0 verschiedenen exit-Status (nicht erfolgreich), wenn **EOF** gelesen wird.

Das Kommando **read** wird üblicherweise für interaktive Eingaben an ein Shell-Skript verwendet.

Beispiel

```
$ cat address␣
echo "Name Vorname"
read name vorname
echo "In welcher Strasse wohnt Herr $name ?"
read strasse
echo "PLZ Wohnort von Herrn $name ?"
read plz wohnort
echo "$name, $vorname, $strasse, $plz $wohnort" >>adresse.txt
sort adresse.txt >adresse.sort
mv adresse.sort adresse.txt
$ chmod u+x address␣
$ address␣
Name Vorname
Meyer Hans␣
In welcher Strasse wohnt Herr Meyer ?
Sandstr. 7␣
PLZ Wohnort von Herrn Meyer ?
90471 Nuernberg␣
$ address␣
Name Vorname
Aller Franz␣
In welcher Strasse wohnt Herr Aller ?
Weinstr. 123␣
PLZ Wohnort von Herrn Aller ?
91058 Erlangen␣
$ cat adresse.txt␣
```

-
1. Eine Eingabezeile kann sich dabei über mehrere Zeilen erstrecken, wenn als letztes Zeichen einer Teilzeile das Zeichen \ vor dem Neuezeile-Zeichen angegeben wird.
 2. Siehe Kapitel 4.6.2.

```
Aller, Franz, Weinstr. 123, 91058 Erlangen
Meyer, Hans, Sandstr. 7, 90471 Nuernberg
$ cat demo↵
echo "Gib eine Zeile von Text ein"
read x
echo $x
echo "Gib noch eine Textzeile ein"
read x y z
echo $x
echo $y
echo $z
$ chmod u+x demo↵
$ demo↵
Gib eine Zeile von Text ein
Das ist eine Eingabe zum Testen von read↵
Das ist eine Eingabe zum Testen von read
Gib noch eine Textzeile ein
Noch ne Eingabe zum Testen von read↵
Noch
ne
Eingabe zum Testen von read
$
```

Hinweis

Bei der Eingabe für ein **read**-Kommando können Metazeichen der Shell durch Voranstellen von `\` ausgeschaltet werden. Der Backslash wird allerdings entfernt, bevor die entsprechenden Worte den angegebenen *variable(n)* zugewiesen werden.

Beispiel

```
$ read satz1 satz2↵
Das Haus, das rot ist↵
$ echo $satz1↵
Das
$ echo $satz2↵
Haus, das rot ist
$ read satz1 satz2↵
Das\ Haus, das rot ist↵
$ echo $satz1↵
Das Haus,
$ echo $satz2↵
das rot ist
$
```

[\`\` schaltet die Leerzeichen-Sonderbedeutung]
[(Worttrenner) aus]

Vordefinierte Shell-Variablen

Neben den frei wählbaren Variablen bietet die Shell auch eine Reihe von Variablen an, deren Namen von ihr bereits fest vorgegeben sind. Bei diesen vordefinierten Variablen ist dann noch zu unterscheiden zwischen

- ▶ vom Benutzer veränderbaren Shell-Variablen und
- ▶ Variablen, die ständig von der Shell automatisch gesetzt werden (auch automatische Variablen genannt).

Vordefinierte, aber änderbare Shell-Variablen

Einigen dieser Variablen weist die Login-Shell während des Anmelde-Vorgangs einen sogenannten default-Wert zu; ist der Benutzer mit diesem voreingestellten Wert nicht zufrieden, kann er ihnen einen neuen Wert zuweisen.

Andere Variablen wiederum werden zwar von der Shell nicht vorbesetzt, können aber vom Benutzer gesetzt werden, um eine bestimmte Konfiguration der Shell festzulegen.

Die Bourne-Shell bietet die folgenden vordefinierten Shell-Variablen an:

Name	Bedeutung
HOME	enthält für den entsprechenden Benutzer den Pfadnamen des Home-Directorys. Wird das Kommando cd ohne Angabe von Argumenten aufgerufen, so wird das Home-Directory das neue Working-Directory (cd entspricht also cd \$HOME). <i>Hinweis:</i> Viele UNIX-Dienstprogramme suchen im Home-Directory nach Konfigurationsdateien, wie z.B. mailx nach der Datei <i>.mailrc</i>. <i>Voreinstellung:</i> Wird automatisch auf einen vom Systemadministrator festgelegten Pfadnamen gesetzt.
PATH	enthält Suchpfade für Programme: Dies sind Directories, in denen beim Aufruf eines Programms oder Shell-Skripts nach der zugehörigen Programm-/Kommandodatei gesucht wird. Die einzelnen Directories sind dabei in der gewünschten Such-Reihenfolge anzugeben und mit : voneinander zu trennen. Ein leeres Directory steht dabei für das Working-Directory. Die Voreinstellung ist meist: PATH=/bin:/usr/bin: (entspricht PATH=/bin:/usr/bin:.)^a Soll neben diesen voreingestellten Such-Directories bei einem Aufruf noch in anderen Directories nach einem Programm oder Shell-Skript gesucht werden, so muß PATH entsprechend gesetzt werden: z.B. PATH=/bin:/usr/bin::/user1/egon:/user1/egon/shellueb Eleganter wäre allerdings in diesem Fall die Zuweisung PATH=\$PATH:\$HOME:\$HOME/shellueb die hier das gleiche bewirken würde. Üblicherweise wird PATH in der Datei <i>.profile</i> gesetzt. <i>Beispiel:</i> Das nachfolgende Shell-Skript <i>which1</i> liefert den Pfadnamen des Programms bzw. Shell-Skripts, welches von der Shell bei alleiniger Angabe des Basisnamens (nicht als absoluter Pfadname) aufgerufen würde:

Tabelle 4.2: Vordefinierte, aber änderbare Shell-Variablen

Name	Bedeutung
	<pre>\$ PATH=./bin:/usr/bin:\$HOME/shellueb ↵ \$ echo \$PATH ↵ ./bin:/usr/bin:/usr1/egon/shellueb \$ cat which1 ↵ find `echo \$PATH   tr ":" " "` -name "\$1" -print line \$ chmod u+x which1 ↵ \$ which1 ls ↵ /bin/ls \$ which1 cph ↵ ./cph \$ which csplit ↵ /usr/bin/csplit \$</pre> Dieses Skript which1 funktioniert allerdings nur, wenn das Working-Directory explizit (mit .) in der PATH-Variablen angegeben wird. An späterer Stelle wird eine verbesserte Version vorgestellt.
CDPATH	enthält die Suchpfade für das builtin-Kommando cd. Wird cd mit einem relativen Pfadnamen aufgerufen, so wird in den Directories, die in CDPATH angegeben sind, nach einem entsprechenden relativen Pfadnamen gesucht. Wird ein solcher gefunden, so wird dies das neue Working-Directory. Die einzelnen Such-Directories sind dabei – wie bei PATH – mit : voneinander zu trennen. <i>Keine Voreinstellung.</i>
MAIL	enthält den Pfadnamen der mailbox-Datei. Jedesmal, wenn neue mail (elektronische Post) in dieser mailbox-Datei ankommt, meldet die Shell dies dem Benutzer (mit <i>you have mail</i>). Dies geschieht allerdings nur dann, wenn die Variable MAILPATH (siehe unten) nicht gesetzt ist. Üblicherweise wird diese Variable in der Datei <i>.profile</i> gesetzt; z.B. mit: <pre>MAIL=/usr/mail/egon</pre>
MAIL-PATH	ist sehr ähnlich zur Variablen MAIL, außer daß hier eine Liste von mailbox-Dateien angegeben werden kann, welche auf Ankunft neuer mail zu überprüfen sind. Die einzelnen mailbox-Pfadnamen sind bei der Angabe mit : zu trennen. Zusätzlich kann zu jedem Pfadnamen noch ein %, gefolgt von einer Nachricht, welche im Falle neuer mail zu melden ist, angegeben werden. Die voreingestellte Meldung ist: <i>you have mail</i>. Z.B. würde die Zuweisung <pre>MAILPATH=/usr/mail/egon:/usr/mail/gruppe%"Postbote war da"</pre> bewirken, daß die Ankunft neuer mail in der mailbox <i>/usr/mail/egon</i> mit »<i>you have mail</i>« und die Ankunft neuer mail in mailbox <i>/usr/mail/gruppe</i> mit »<i>Postbote war da</i>« gemeldet würde. <i>Keine Voreinstellung.</i>

Tabelle 4.2: Vordefinierte, aber änderbare Shell-Variablen (Fortsetzung)

Name	Bedeutung
MAIL-CHECK	legt das Zeitintervall (in Sekunden) fest, in dem ständig zu überprüfen ist, ob neue mail in den über MAILPATH oder MAIL (nur, wenn MAILPATH nicht gesetzt) festgelegten mailbox-Dateien angekommen ist. Wenn MAILCHECK mit 0 besetzt ist, dann prüft die Shell vor jeder Ausgabe des Primär-Promptzeichens auf Ankunft neuer mail. Voreinstellung: MAILCHECK=600 (entspricht 10 Minuten).
PS1	enthält das Primär-Promptzeichen bzw. den Primär-Promptstring, den die Shell ausgibt, wenn sie die Eingabe von Kommandos erwartet. Der Promptstring kann allerdings nur statisch und nicht dynamisch – wie z.B. unter MS-DOS – festgelegt werden. Beispiel: <pre>\$ PS1="Gib ein, egon> " Gib ein, egon> PS1="`pwd`> " /user1/egon/shellueb> cd .. /user1/egon/shellueb> PS1="\$ " \$ cd shellueb \$</pre> Hieraus ist zu ersehen, daß die Änderung des Working-Directories nicht im Promptstring berücksichtigt wurde, da die Kommandosubstitution `pwd` bei der Zuweisung an PS1 bereits ausgewertet und der von ihr gelieferte Pfadname (und nicht die Kommandosubstitution selbst) der Variablen PS1 zugewiesen wurde. Voreinstellung: PS1="\$ ".
PS2	enthält den Sekundär-Promptstring, welchen die Shell ausgibt, wenn sich eine Kommandozeile über mehrere Zeilen erstreckt, um anzuzeigen, daß sie noch auf weitere Eingaben wartet, bevor sie mit der Ausführung der gesamten Kommandozeile beginnt. Beispiele: <pre>\$ PS2="wie gehts weiter? " \$ echo "Das ist wie gehts weiter? eine wie gehts weiter? Kommando wie gehts weiter? zeile" Das ist eine Kommandozeile \$ cat wh\ wie gehts weiter? ich1 find `echo \$PATH` tr ":" " " -name "\$1" -print line \$ PS2="> " \$</pre> Voreinstellung: PS2="> ".

Tabelle 4.2: Vordefinierte, aber änderbare Shell-Variablen (Fortsetzung)

Name	Bedeutung
IFS	(<i>Internal Field Separators</i>) enthält die Trennzeichen, welche von der Shell zum Trennen von einzelnen Wörtern auf der Kommandozeile oder in einer Eingabezeile (beim Kommando read) verwendet werden. Beispiel: <pre>\$ echo \$IFS [Leer-, Tab- und Neuzeile-Zeichen] [werden hier ausgegeben] \$ ls s* w* schiebe stdio.h which1 \$ ls s*,w* s*,w*: No such file or directory \$ ALT_IFS="\$IFS" \$ IFS="," \$ ls s*,w* schiebe stdio.h which1 \$ IFS=\$ALT_IFS \$</pre> Voreinstellung ist: Leerzeichen, Tabulatorzeichen und Neuezeile-Zeichen.
TERM	enthält die Bezeichnung des Terminals, an welchem der Benutzer momentan arbeitet. Bildschirmorientierte Programme wie z.B. vi oder pg, benutzen den Inhalt dieser Shell-Variablen, um die richtigen Bildsteuerungs-Sequenzen abzusetzen. Keine Voreinstellung.
LOGNAME	enthält den Login-Namen des jeweiligen Benutzers. Der Aufruf echo \$LOGNAME entspricht dem Aufruf des Kommandos logname .
SHELL	Wenn eine neue Shell aufgerufen wird, so durchsucht diese ihre Laufzeitumgebung (engl.: <i>environment</i>) nach einer Variablen mit dem Namen SHELL . Existiert diese Variable und enthält sie als Wert einen Pfadnamen, dessen Basisname "rsh" ist, so wird die neue Shell nicht als normale Shell, sondern als eine eingeschränkte Shell gestartet; siehe Kapitel 4.20, das die eingeschränkte Shell rsh beschreibt.
SHACCT	Wenn diese Variable mit den Namen einer vom entsprechenden Benutzer beschreibbaren Datei besetzt ist, dann wird für jedes ausgeführte Shell-Skript Abrechnungs-Information (engl.: <i>accounting record</i>) in diese Datei geschrieben.
TZ	(<i>time zone</i>) enthält Angaben zur Zeitzone. Das Format für die Angabe ist xxxnzzz, wobei xxx die Abkürzung für die lokale Standardzeit,

Tabelle 4.2: Vordefinierte, aber änderbare Shell-Variablen (Fortsetzung)

Name	Bedeutung
<i>n</i>	die Differenz in Stunden zu GMT und
<i>zzz</i>	die Abkürzung für die lokale Sommerzeit ist.

Tabelle 4.2: Vordefinierte, aber änderbare Shell-Variablen (Fortsetzung)

- Zuerst in **/bin** dann in **/usr/bin** und schließlich im Working Directory nach einem ausführbaren Programm oder Shell-Skript suchen.
- Ist der Benutzer der Superuser, so ist die Voreinstellung: **PS1="# "**.
- Siehe »Linux / Unix-Grundlagen«.

Die Shell initialisiert (vergibt Voreinstellungen) an die Parameter **PATH**, **PS1**, **PS2**, **IFS** und **MAILCHECK**.

HOME und **MAIL** werden bei jedem Anmeldevorgang neu gesetzt.

Die momentanen Werte aller Shell-Variablen können mit dem Kommando **set** (ohne Angabe von Argumenten) am Bildschirm aufgelistet werden.

Beispiel `$ set` 

```
ALT_IFS=

HOME=/user1/egon
IFS=
LOGNAME=egon
MAIL=/usr/mail/egon
MAILCHECK=0
OPTIND=1                [wird beim builtin-Kommando getopts besprochen]
PATH=./bin:/usr/bin:/user1/egon/shellueb
PS1=$
PS2=>
TERM=vt100
TZ=GMT0
$
```

Automatische Variablen

Die folgenden vordefinierten Variablen werden ständig neu von der Shell gesetzt, wobei auf die Werte dieser automatischen Parameter wieder durch Vorstellen von **\$** zugegriffen werden kann:

Variablenname	Bedeutung
#	Anzahl der gesetzten Positionsparameter.
-	Optionen, die beim Aufruf der Shell angegeben oder mit dem set -Kommando eingeschaltet wurden ^a .

Tabelle 4.3: Automatische Variablen

Variablenname	Bedeutung
?	Exit-Status des zuletzt im Vordergrund ausgeführten Kommandos.
\$	Prozeßnummer (PID) der aktuellen Shell.
!	Die Prozeßnummer (PID) des zuletzt im Hintergrund gestarteten Kommandos.
*	Alle Positionsparameter als ein String: "\$*" entspricht "\$1 \$2 \$3 ..."
@	Alle Positionsparameter als einzelne Strings: "\$@" entspricht "\$1" "\$2" "\$3" ...

Tabelle 4.3: Automatische Variablen

a. Siehe auch Kapitel 4.15.6.

Beispiel

```
$ echo $$
1307
$ cat argu_pid
echo Es sind $# Argumente uebergeben wurden.
echo Momentane PID: $$
$ chmod u+x argu_pid
$ argu_pid arg1 arg2
Es sind 2 Argumente uebergeben wurden.
Momentane PID: 1529
$ echo $$
1307
$
```

An diesem Beispiel ist sehr schön zu erkennen, daß der Aufruf eines Shell-Skripts eine neue Subshell startet.

Der Unterschied zwischen den beiden automatischen Variablen

- * Alle Positionsparameter als **ein** String: "\$*" entspricht "\$1 \$2 \$3 ..." und
- @ Alle Positionsparameter als **einzelne** Strings: "\$@" entspricht "\$1" "\$2" "\$3" ...

hat besonders innerhalb von Shell-Skripts Bedeutung, wenn in der Kommandozeile angegebene Argumente beim Aufruf weiterer Shell-Skripts weitergegeben werden.

Beispiel

```
$ cat kinder1
echo Level 1:
echo "$# kleine Kinder: $*"
echo "$# kleine Kinder: $@"
echo
kinder2 "$*"
kinder2 "$@"
```

```
$ cat kinder2 ↵
echo "-----"
echo Level 2:
echo "$# kleine Kinder:"
echo Name 1: $1
echo Name 2: $2
echo Name 3: $3
$ chmod u+x kinder1 kinder2 ↵
$ kinder1 franz michel sascha ↵
Level 1:
3 kleine Kinder: franz michel sascha
3 kleine Kinder: franz michel sascha

-----
Level 2:
1 kleine Kinder:
Name 1: franz michel sascha
Name 2:
Name 3:
-----
Level 2:
3 kleine Kinder:
Name 1: franz
Name 2: michel
Name 3: sascha
$
```

4.6.3 Spezielle Variablenausdrücke

Die Shell unterscheidet zwischen undefinierten Variablen und Variablen, denen explizit der »Nullwert« (leere Zeichenkette) mit einer der drei folgenden Anweisungen zugewiesen wurde:

```
name=
name=' '
name=" "
```

Bei den letzten beiden Zuweisungen darf kein Leerzeichen innerhalb von " und "" angegeben sein.

Neben dem einfachen Zugriff auf den Wert einer Variable mit `$variable` bietet die Shell auch einige spezielle Zugriffsmöglichkeiten auf die Werte von Variablen:

`${variable}`

ist identisch zur Angabe **`$variable`**. Allerdings kann diese Zugriffsart auch dazu verwendet werden, um den Wert einer Variablen in eine Zeichenkette einzubetten.

Beispiel

```
$ bezeich=Lehrer↵
$ echo Guten Tag, liebe $bezeichin↵
Guten Tag, liebe
$ echo Guten Tag, liebe ${bezeich}in↵
Guten Tag, liebe Lehrerin
$
```

Im ersten Fall sucht die Shell nach einer Variable **bezeichin**, die sie allerdings nicht finden kann, da eine solche Variable nicht definiert wurde; somit gibt sie den Nullwert aus. Im zweiten Fall dagegen findet die Shell die Variable **bezeich** und gibt deren Inhalt mit dem angehängten String "in" aus.

Diese Zugriffsart auf den Wert einer Variablen muß immer dann verwendet werden, wenn dem angegebenen Variablennamen direkt ein Buchstabe, eine Ziffer oder ein Unterstrich folgt.

Wenn für **variable** eine Ziffer angegeben ist, so wird der Wert des entsprechenden Positionsparameters hierfür eingesetzt.

Beispiel

```
$ cat weiblich↵
echo ${1}in
$ chmod u+x weiblich↵
$ weiblich schueler↵
schuelerin
$ weiblich direktor↵
direktorin
$
```

\${variable:-wort}

Diese Ausdrucksform liefert folgenden Wert:

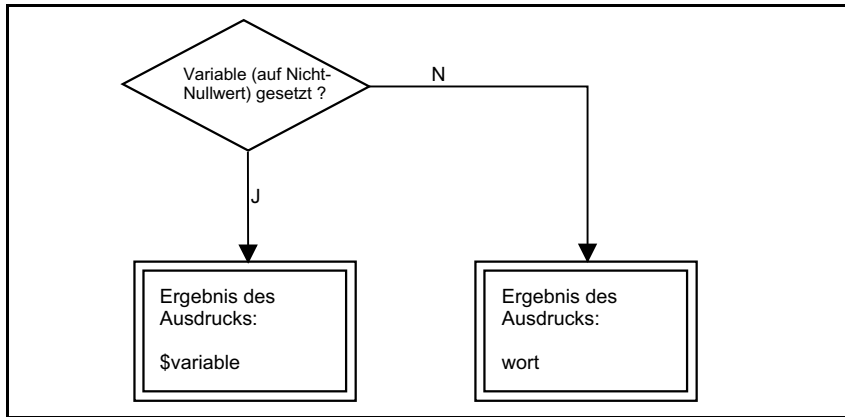
Ergebnis des Ausdrucks::=

```
if variable (mit Nicht-Nullwert) gesetzt
then $variable
else die mit wort angegebene Zeichenkette.
fi
```

Der Ablaufplan in Abbildung 4.8 soll dies veranschaulichen.

Beispiel

```
$ sonne=sonnig↵
$ urlaub=↵
$ echo ${sonne:-regnerisch} und ${urlaub:-viel} zu tun↵
sonne und viel zu tun
$ echo $sonne↵
sonnig
$ echo $urlaub↵
$
```

Abbildung 4.8: Ablaufplan zu `${variable:-wort}`

Da `sonne` gesetzt ist, wird der Wert dieser Variablen ausgegeben, während bei der 2. Konstruktion `urlaub` mit einem Nullwert besetzt ist, und somit wird die alternativ angegebene Zeichenkette "viel" ausgegeben.

```

$ cat woist ↵
dir=${2:-$HOME}
find $dir -name $1 -print
$ chmod u+x woist ↵
$ woist countdown ↵
/user1/egon/shellueb/countdown
$ woist calendar ↵
/usr/bin/calendar
$
  
```

`${variable-wort}`

Diese Ausdrucksform (Doppelpunkt fehlt) liefert folgenden Wert:

Ergebnis des Ausdrucks::=

```

if variable definiert1
then $variable
else die mit wort angegebene Zeichenkette.
fi
  
```

Beispiel

```

$ urlaub= ↵
$ echo ${urlaub:-viel} ↵ [urlaub ist mit Nullwert besetzt]
viel
$ echo ${url:-viel} ↵ [url ist nicht definiert (niemals initialisiert)]
viel
  
```

1. Eventuell auch mit Nullwert gesetzt.

```
$ echo ${urlaub-viel} [urlaub ist mit Nullwert besetzt]
                        [Ausgabe der leeren Zeichenkette von urlaub]
$ echo ${url-viel}    [url ist nicht definiert (niemals initialisiert)]
viel
$
```

Dieses Beispiel verdeutlicht den Unterschied zwischen einer undefinierten und einer explizit mit dem Nullwert vorbesetzten Variablen.

`${variable:=wort}`

Diese Ausdrucksform entspricht folgendem Pseudocode:

Ergebnis des Ausdrucks::=

```
if variable nicht gesetzt ist oder aber Nullwert enthält
then variable=wort
fi
$variable
```

Der Ablaufplan in Abbildung 4.9 soll dies nochmals veranschaulichen.

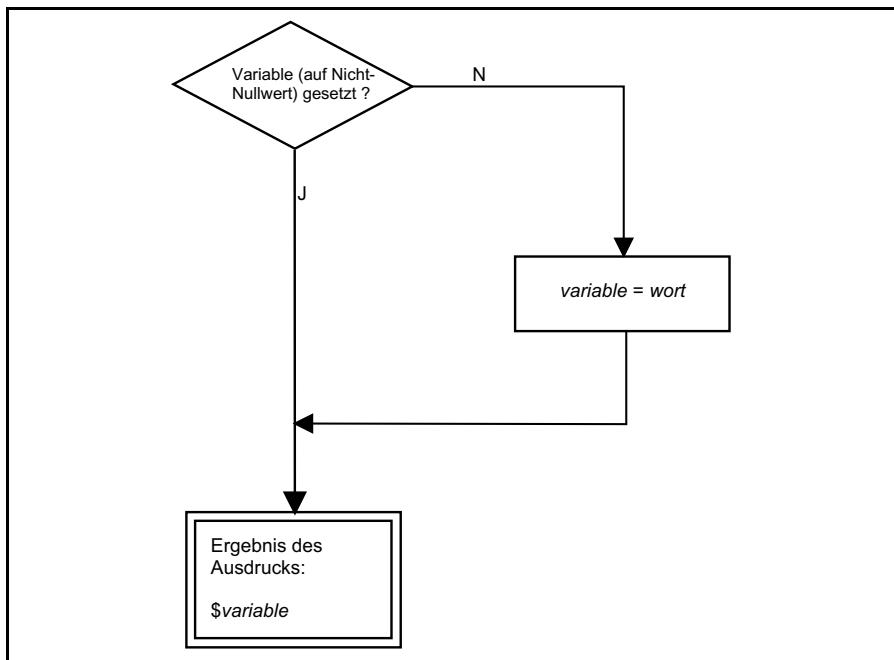


Abbildung 4.9: Ablaufplan zu `${variable:=wort}`

Beispiel

```
$ sonne=sonnig
$ urlaub=
$ echo ${sonne:=regnerisch} und ${urlaub:=viel} zu arbeiten
sonnig und viel zu arbeiten
$ echo $sonne
sonnig
$ echo $urlaub
viel
$
```

Diese Konstruktion kann nicht verwendet werden, um Positionsparametern Werte zuzuweisen.

`${variable=wort}`

Diese Ausdrucksform (Doppelpunkt fehlt) entspricht folgendem Pseudocode:

Ergebnis des Ausdrucks::=

```
if variable nicht definiert
then variable=wort
fi
$variable
```

`${variable:?wort}`

Diese Ausdrucksform entspricht folgendem Pseudocode:

Ergebnis des Ausdrucks::=

```
if variable (mit Nicht-Nullwert) gesetzt
then $variable
else
  if wort angegeben
  then wort ausgeben
  else "parameter null or not set" ausgeben
fi
Shell-Skript verlassen
fi
```

Der Ablaufplan von Abbildung 4.10 soll dies veranschaulichen.

Beispiel

```
$ sonne=sonnig
$ echo ${sonne:? "Oh du schoene Urlaubszeit"}
sonnig
$ sonne=
$ echo ${sonne:? "Hallo Regentroepfchen"}
sonne: Hallo Regentroepfchen
$ echo ${sonne:?}
sonne: parameter null or not set
$
```

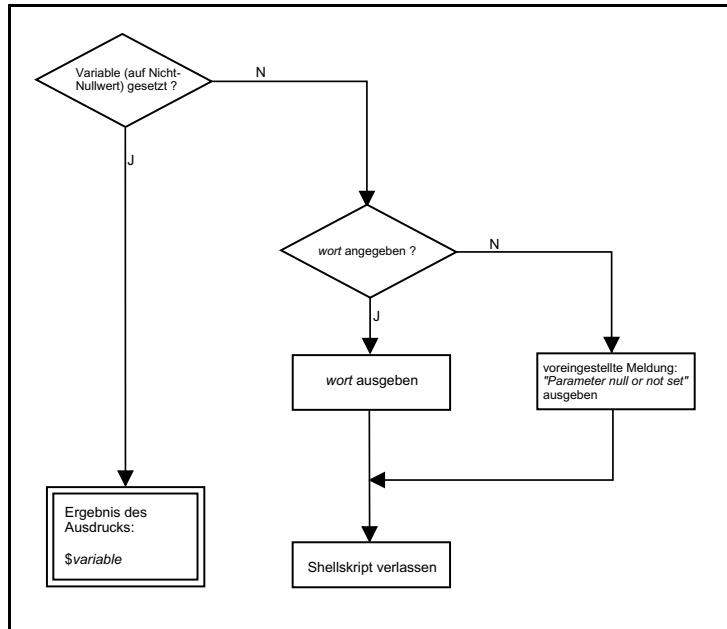


Abbildung 4.10: Ablaufplan zu `${variable:?word}`

`${variable?word}`

Fehlt der Doppelpunkt bei der Angabe, so ändert sich lediglich die erste Abfrage im Pseudocode:

Ergebnis des Ausdrucks::=

```

if variable definiert
then $variable
else
  if word angegeben
  then word ausgeben
  else "parameter null or not set" ausgeben
fi
Shell-Skript verlassen
fi
  
```

`${variable:+word}`

Diese Ausdrucksform entspricht folgendem Pseudocode:

Ergebnis des Ausdrucks::=

```

if variable (mit Nicht-Nullwert) gesetzt
then word
else Nullwert
fi
  
```

Der Ablaufplan von Abbildung 4.11 soll dies veranschaulichen.

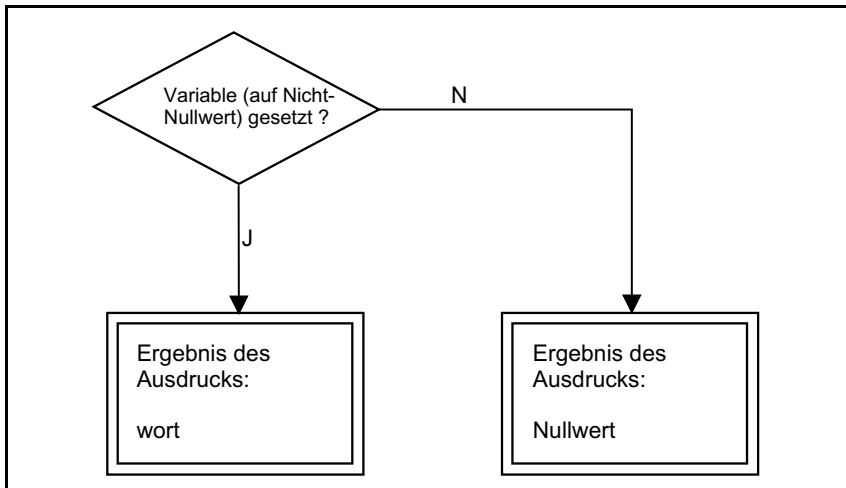


Abbildung 4.11: Ablaufplan zu `${variable:=wort}`

Beispiel

```
echo ${TEST:+"Der Wert von i ist" $i}
```

Ein Shell-Skript könnte mit solchen Anweisungen »gespickt« sein. Während der Testphase wird die Variable **TEST** gesetzt und alle Werte von *i* zum Nachvollziehen am Bildschirm ausgegeben. Ist dieses Skript ausgetestet, kann die Variable **TEST** mit dem Nullwert (**TEST=**) vorbesetzt werden. Es erfolgt keine Testausgabe mehr.

`${variable+wort}`

Fehlt der Doppelpunkt bei der Angabe, so ändert sich lediglich die erste Abfrage im Pseudocode:

Ergebnis des Ausdrucks::**=**

```
if variable definiert
then wort
else Nullwert
fi
```

In all diesen angegebenen Variablenausdrücken kann **wort** eine einfache Zeichenkette (String) oder ein Ausdruck sein, der nach seiner Auswertung eine Zeichenkette liefert.

Beispiel

```
$ dinner="Schweinebraten"
$ echo ${essen:=="${dinner} mit Salat"}
Schweinebraten mit Salat
$ echo $essen
Schweinebraten mit Salat
```



```
$ echo ${name:=`logname`}
egon
$ echo $name
egon
$ cat gls
dir=${1:-`pwd`}
para2=$2
ende=$3
tmp_name=/tmp/${4:-`logname`}.$$ # eindeutigen Namen fuer
                                # eine temporaere Datei
                                # festlegen; dazu wird
                                # der eigene Loginname
                                # (wenn $4 leer ist) und
                                # die PID ($$) verwendet

trenner=${para2:="+++++"}
banner $LOGNAME >$tmp_name
echo $dir >>$tmp_name
echo $trenner >>$tmp_name
ls -CF $dir >>$tmp_name
echo $trenner >>$tmp_name
cat $tmp_name
rm -r $tmp_name
echo ${ende:?"Ich bin fertig"}
$ chmod u+x gls
$ gls . ----- Tschuess

#####  #####  #####  #  #
#      #  #  #  #  ##  #
#####  #      #  #  #  #
#      #  ###  #  #  #  #
#      #  #  #  #  #  ##
#####  #####  #####  #  #

.
-----
adress*      countdow2*  div_null.c  kinder2*      which1*
adresse.txt  countdown*  div_null2*  ll*           woist*
anfang*      cph*         eing_zaehl  lszaehl       zaehle.txt
argu_pid*    datum*       gls*        schiebe*      zeig_pid*
ausgab*      demo*        gruss*      stdio.h
core         div_null*    kinder1*    weiblich*

-----
Tschuess
$ gls /usr/include

#####  #####  #####  #  #
#      #  #  #  #  ##  #
#####  #      #  #  #  #
#      #  ###  #  #  #  #
#      #  #  #  #  #  ##
#####  #####  #####  #  #
```

```

/usr/include
+++++++
a.out.h      fatal.h      mnttab.h      search.h      term.h
agent.h      fcntl.h      mon.h         setjmp.h      termio.h
alarm.h      filehdr.h    nan.h         sgtty.h       time.h
aouthdr.h    ftw.h        nlist.h       signal.h      tiuser.h
ar.h         grp.h        nsaddr.h      stand.h       tp_defs.h
assert.h     ieeeep.h     nserve.h      stdio.h       ttysrv.h
core.h       ldfcn.h      pn.h          storclass.h   unctrl.h
ctype.h      limits.h     poll.h        string.h      unistd.h
curses.h     linenum.h    prof.h        stropts.h     ustat.h
dial.h       macros.h     pwd.h         strselect.h   utmp.h
dirent.h     malloc.h     regexp.h      symbol.h      values.h
dumprestor.h math.h       reloc.h       syms.h        values.h3b5x
errno.h      math.h3b5x   rje.h         sys/          varargs.h
execargs.h   memory.h     scnhdr.h      sys.s
+++++++
gls: ende: Ich bin fertig
$

```

Es ist noch darauf hinzuweisen, daß **wort** nur in den Fällen ausgewertet wird, in denen sein Wert auch verwendet wird.

Beispiel `echo ${1:-`cd $HOME;pwd`}`

Wurde eine solche Anweisung in einem Shell-Skript gegeben, dann ist im weiteren Verlauf des Shell-Skripts davon auszugehen, daß möglicherweise der Directory-Wechsel (**cd \$HOME**) nicht durchgeführt wurde (wenn **\$1** auf einen Nicht-Nullwert gesetzt war).

Hinweis Allgemein gilt folgendes: Wird bei einer dieser Angaben der Doppelpunkt nicht angegeben, so wird nicht der Inhalt der entsprechenden Variablen geprüft, sondern ob diese Variable bereits definiert ist. Eine Variable gilt als definiert, wenn sie zuvor explizit gesetzt wurde, eventuell auch mit einer leeren Zeichenkette.

Typische Anwendungen:

Da diese speziellen Variablenausdrücke sehr schwer zu merken sind, werden hier typische Anwendungen zu den einzelnen Konstruktionen gegeben, um dem Benutzer kleine Merkhilfen zu geben:

`dir=${2:-.}` bzw. `dir=${2-..}`

wird häufig innerhalb von Shell-Skripts verwendet, um Variablen entweder einen vom Skript-Aufrufer angegebenen Wert oder einen default-Wert zuzuweisen.

Wurde hier beim Aufruf des Shell-Skripts das 2.Argument angegeben, so wird dieses Argument (Directory) der Variablen *dir* zugewiesen, ansonsten wird dieser Variablen der default-Wert (hier das Working-Directory) zugewiesen.

```
${EDITOR:=/bin/vi} bzw. ${EDITOR=/bin/vi}
```

wird auch sehr häufig in Shell-Skripts verwendet, um sicherzustellen, daß bestimmte Variablen im weiteren Verlauf in jedem Fall mit einem Wert besetzt sind.

In diesem Beispiel wird die Variable *EDITOR* nur dann auf */bin/vi* gesetzt, wenn sie noch leer (bzw. undefiniert) ist.

```
${1:? "1.Argument muss immer angegeben sein"}  
bzw.: ${1:? "1.Argument muss immer angegeben sein"}
```

kann verwendet werden, um zu prüfen, ob Variablen gesetzt (bzw. definiert) sind, die für die Weiterarbeit in Shell-Skripts von Wichtigkeit sind. Wenn nicht, so wird das entsprechende Shell-Skript mit der angegebenen Fehlermeldung abgebrochen.

```
cd ${WECHSEL:+"/user1/egon/shellueb"}  
bzw.: cd ${WECHSEL+"/user1/egon/shellueb"}
```

kann verwendet werden, um eine **if**-Anweisung nachzubilden.

Wenn in diesem Beispiel die Variable *WECHSEL* gesetzt (bzw. definiert) ist, so wird zum Directory */user1/egon/shellueb* und ansonsten zum Home-Directory gewechselt.

Diese Form eines Variablenausdrucks wird allerdings nicht allzuoft in der Praxis verwendet.

4.6.4 Gültigkeitsbereiche von Variablen

Neben den Argumenten beim Start einer Subshell stellt die Elternhell der entsprechenden Subshell darüberhinaus eine Umgebung (engl.: *environment*) zur Verfügung. Eine solche Umgebung enthält eine Liste von Namen mit zugehörigen Werten.

Der Inhalt für die Umgebung der Login-Shell wird bereits beim Anmelden eines Benutzers festgelegt: Beim Aufruf der Login-Shell wird die vom System vorgegebene Standardumgebung gelesen und für jedes gefundene Name/Werte-Paar wird eine Shell-Variable mit dem angegebenen Namen und Wert erzeugt. Diese Umgebung der Login-Shell kann durch Definitionen in der Datei *.profile* für die speziellen Bedürfnisse des jeweiligen Benutzers angepaßt werden. So könnte z.B. in *.profile*

```
PS1="Was nun> "
```

angegeben sein. Dadurch würde der voreingestellte Wert von **PS1** (" ") auf **"Was nun> "** geändert werden.

Beim Start einer Subshell liest die gerade aktive Shell nun die Standardumgebung (nicht die Umgebung der Elternshell) und kreiert für jedes dort angegebene Name/Werte-Paar eine Shell-Variable, die dann dieser Subshell zur Verfügung steht.

Wird nun von einer Subshell eine aus der Standardumgebung bereitgestellte Shell-Variable verändert oder eine neue eingeführt, so hat dies nur Auswirkungen auf die Kopie, also auf die eigene lokale Umgebung des jeweiligen Programms und nicht auf die Umgebung der Elternshell.

Würde z.B. in der Subshell die neue Variable **insekt** eingeführt

`insekt="fliege"`

hätte dies nur Auswirkungen auf die lokale Umgebung (siehe auch Abbildung 4.12):

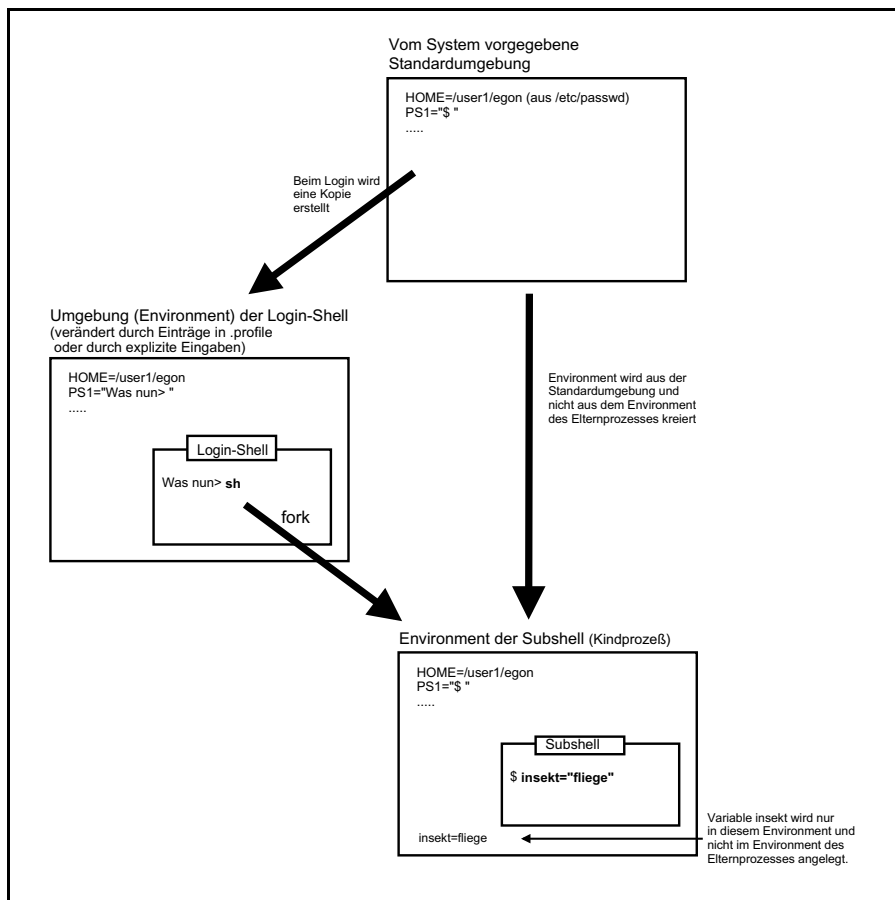


Abbildung 4.12: Environment einer Subshell

```
Beispiel $ PS1="Gib ein, `logname`> " [Setzen eines neuen Primär-Promptstrings]
Gib ein, egon> NACHN=Meier [Zuweisen eines Werts an Variable NACHN]
Gib ein, egon> set [Ausgeben aller gesetzten Shell-Variablen]
HOME=/usr1/egon
IFS=

LOGNAME=egon
MAIL=/usr/mail/egon
MAILCHECK=0
NACHN=Meier
OPTIND=1
PATH=:/bin:/usr/bin
PS1=Gib ein, egon>
PS2=>
TERM=vt100
TZ=GMT0
Gib ein, egon> sh [Starten einer neuen Subshell]
$ echo $NACHN [Primär-Prompt aus Standardumgebung (nicht
der zuvor definierte)]
[Variable NACHN hier (in Subshell) nicht
definiert]

$ set [Ausgeben aller (in Subshell) gesetzten
Shell-Variablen]
HOME=/usr1/egon
IFS=

LOGNAME=egon
MAIL=/usr/mail/egon
MAILCHECK=600
OPTIND=1
PATH=:/bin:/usr/bin
PS1=$ [Primär-Promptstring aus Standardumgebung
(nicht der aus Elternshell)]
PS2=>
TERM=vt100
TZ=GMT0
$ exit
Gib ein, egon> echo $NACHN [NACHN in dieser Shellebene wieder definiert]
Meier
Gib ein, egon> PS1="$ "
$
```

Aus diesem Beispiel ist zu ersehen, daß mit dem Start eines Shell-Skripts, welches von einer Subshell ausgeführt wird, die Veränderung von **PS1** und die Neudefinition von **NACHN** in dieser neuen Subshell unbekannt sind.

Bei einer Rückkehr in die aufrufende Shell stehen jedoch die zuvor dort neudefinierten oder veränderten Werte wieder zur Verfügung, da diese lokale Umgebung der aufrufenden Shell den Aufruf einer Subshell »überlebt«.

Exportieren von Shell-Variablen

Nun ist es manchmal auch erwünscht, veränderte oder neu eingeführte Variablen an eine Subshell zu vererben. Dazu steht das builtin-Kommando **export** zur Verfügung:

export variable [variable ...]

Beispiel

```
$ PS1="Gib ein, `logname`> "
Gib ein, egon> NACHN=Meier
Gib ein, egon> export PS1 NACHN [Markiert Var. PS1 und NACHN für Export]
Gib ein, egon> sh [Starten einer neuen Subshell]
Gib ein, egon> echo $NACHN [Verändert. Prompt nun auch in Subshell vorh.]
Meier [Variable NACHN hier (in Subshell) nun definiert]
Gib ein, egon> set [Ausgeben aller (in Subshell) gesetzten
Shell-Variablen]

HOME=/user1/egon
IFS=
LOGNAME=egon
MAIL=/usr/mail/egon
MAILCHECK=600
NACHN=Meier [Variable NACHN hier (in Subshell) nun definiert]
OPTIND=1
PATH=:bin:/usr/bin
PS1=Gib ein, egon> [Veränderter Promptstring nun auch in Subshell
vorhanden]

PS2=>
TERM=vt100
TZ=GMT0
Gib ein, egon> PS1="Next please> "
Next please> NACHN=MueLLer
Next please> set
HOME=/user1/egon
IFS=

LOGNAME=egon
MAIL=/usr/mail/egon
MAILCHECK=600
NACHN=MueLLer
OPTIND=1
PATH=:bin:/usr/bin
PS1=Next please>
PS2=>
TERM=vt100
TZ=GMT0
Next please> exit [Verlassen der Subshell]
Gib ein, egon> echo $NACHN
Meier [NACHN besitzt den alten Wert, nicht den Wert
"Mueller"]

Gib ein, egon> set
HOME=/user1/egon
```

```
IFS=

LOGNAME=egon
MAIL=/usr/mail/egon
MAILCHECK=0
NACHN=Meier
OPTIND=1
PATH=:/bin:/usr/bin
PS1=Gib ein, egon>      [alter Promptstring; nicht von Subshell verändert]
PS2=>
TERM=vt100
TZ=GMT0
Gib ein, egon> PS1="$ " 
$
```

export bewirkt, daß die hierbei angegebenen Variablen an alle Subshells, Sub-Subshells usw. der momentan aktuellen Shell weitergereicht werden. Am leichtesten läßt sich dies vielleicht so vorstellen, daß beim Start einer Subshell zuerst alle **exportierten** Variablen in die Umgebung dieser Subshell übernommen werden, bevor dann die restlichen Shell-Variablen aus der Standardumgebung gebildet werden.

Wenn allerdings in einer solchen Subshell eine **exportierte** Variable modifiziert wird, dann hat dies keine Auswirkungen auf den Wert dieser Variablen in der Elternshell, da eine solche Änderung wieder nur in der lokalen Umgebung der jeweiligen Subshell vorgenommen wird.

Der Aufruf

set -a

bewirkt im übrigen fast das gleiche wie das Kommando **export**. Der Unterschied besteht darin, daß nur die ab diesem Zeitpunkt neu angelegten oder veränderten Variablen exportiert werden:

```
$ VORN=Helmut 
$ sh 
$ echo $VORN 

$ exit 
$ set -a 
$ sh 
$ echo $VORN 

$ exit 
$ VORN=Manfred 
$ sh 
$ echo $VORN 
Manfred
$ exit 
$
```

Löschen von Shell-Variablen

Mit dem builtin-Kommando **unset** kann eine Shell-Variable aus der lokalen Umgebung einer Subshell entfernt werden. Dies kann bei umfangreichen Shell-Skripts sinnvoll sein, um den Speicherplatz von nicht mehr benötigten Shell-Variablen wieder frei zu geben. Wird in einer Subshell (S) eine **exportierte** Variable, die von der Elternshell (V) geerbt wurde, mit **unset** aus der lokalen Umgebung entfernt, so steht diese auch möglichen weiteren Subshells zur momentan aktuellen Shell (S) nicht mehr zur Verfügung.

Ausgabe aller exportierten Shell-Variablen

Der Aufruf von

```
export    [ohne Angabe von Argumenten]
```

bewirkt, daß alle Variablen, die während der momentanen UNIX-Sitzung als »zu exportieren« markiert wurden, ausgegeben werden. Dazu zählen auch die Variablen, die z.B. in *.profile* mit **export** exportiert wurden. Nicht aufgelistet werden dagegen in diesem Fall die Variablen, die automatisch während der Login-Prozedur (wie z.B.: **HOME**) exportiert wurden:

```
$ export ↵
export NACHN
export PATH
export PS1
export TERM
export TZ
export VORN
$
```

Der Aufruf des Kommandos **env** ohne Argumente gibt **alle** momentan **exportierten** Variablen mit deren aktuellem Wert aus.

Beispiel

```
$ env ↵
HOME=/user1/egon
LOGNAME=egon
MAIL=/usr/mail/egon
NACHN=Meier
PATH=./bin:/usr/bin
PS1=$
TERM=vt100
TZ=GMT0
VORN=Manfred
$
```


Übergabe von Shellvariablen an Subshells ohne export

Des öfteren benötigt man für die Ausführung eines einfachen Kommandos zusätzliche Shell-Variablen, wobei allerdings die ursprüngliche Shell-Umgebung nicht eigens hierfür verändert werden soll. Dieser Anforderung kann leicht Genüge getan werden, indem beim Aufruf vor dem entsprechenden Programm- oder Skript-Namen die geeigneten Variablen-Definitionen mit-angegeben werden.

Beispiel `var_wert` sei ein Shell-Skript, welches auf die Variablen **HOME**, **LEVEL** und **PATH** zugreift:

```
$ cat var_wert
echo "LEVEL=$LEVEL"
echo "HOME=$HOME"
echo "PATH=$PATH"
$ chmod u+x var_wert
$ var_wert
LEVEL=
HOME=/user1/egon
PATH=:bin:/usr/bin
$ LEVEL=2 HOME=/user1/fritz PATH=$HOME/bin:/bin: var_wert
LEVEL=2
HOME=/user1/fritz
PATH=/user1/egon/bin:/bin:
$ echo $PATH
:/bin:/usr/bin
$ echo $HOME
/user1/egon
$ echo $LEVEL

$
```

[Ursprüngl. Werte der 3 Var. mit vorherig. Aufruf nicht verändert]

Neben der erwähnten Option **-a** bietet das built-in Kommando **set** noch weitere Optionen an, z.B. die Option **-k**, welche bewirkt, daß alle Shell-Variablen-Definitionen an die Umgebung einer Subshell weitergegeben werden, sogar wenn diese nach dem Kommando-/Shell-Skript-Namen angegeben werden. **+k** schaltet diesen Mechanismus wieder aus.

Beispiel

```
$ var_wert LEVEL=2 HOME=/user1/fritz PATH=$HOME/bin:/bin:
LEVEL=
HOME=/user1/egon
PATH=:bin:/usr/bin
$ set -k
$ var_wert LEVEL=2 HOME=/user1/fritz PATH=$HOME/bin:/bin:
LEVEL=2
HOME=/user1/fritz
```

[Variablenangaben hier als Positionsparam. übergeben ---> keine Zuweisungen]

[Nun Zuweisungen von Werten an Variablen (keine Argumente für var_wert)]

```
PATH=/user1/egon/bin:/bin:
$ set +k [↵] [Ursprüngl. Einstellung wiederherstellen]
$ echo dach=ziegel steinhaus [↵] [2 Argumente fuer echo]
dach=ziegel steinhaus
$ set -k [↵]
$ echo dach=ziegel steinhaus [↵] [steinhaus Argument; dach=ziegel Zuweisung]
steinhaus
$ set +k [↵] [Ursprüngl. Einstellung wiederherstellen]
$
```

Hinweise

Shell-Variablen einer Subshell können niemals an eine Elternshell zurückgegeben werden, da jede Subshell eine Kopie von der Umgebung ihres Elternprozesses erhält. Alle Änderungen (einschließlich **unset**) wirken sich auf diese Kopie und nicht auf die Standardumgebung oder auf die Umgebung einer Elternshell aus. Deswegen kann ein Shell-Skript nicht verwendet werden, um eine Shell-Variable in der Elternshell zu kreieren und zu modifizieren.

Da eine Subshell ihre eigene lokale Umgebung besitzt, zu der beispielsweise auch das Working-Directory für diese Shell gehört, ist es eine typische Anwendung, daß ein kurzfristiger Wechsel in ein anderes Directory in einer Subshell vorgenommen wird, um sich dann mit dem Verlassen dieser Subshell sofort wieder im ursprünglichen Directory zu befinden:

```
$ pwd [↵]
/user1/egon/shellueb
$ sh [↵]
$ cd /usr/include/sys [↵]
$ pwd [↵]
/usr/include/sys
$ ls d* [↵]
debug.h
debugreg.h
dir.h
dirent.h
dma.h
$ exit [↵]
$ pwd [↵]
/user1/egon/shellueb
$
```

Die C-Shell bietet – wie in Kapitel 6 gezeigt wird – für solche kurzfristigen Directorywechsel eigene Kommandos (**pushd** und **popd**) an.

4.7 Expandierung von Dateinamen auf der Kommandozeile

Beim Aufruf eines Kommandos oder Shell-Skripts wird jedes Wort der Kommandozeile von der Shell daraufhin untersucht, ob eines der Zeichen *, ? oder [darin vorkommt. Wird ein solches Wort gefunden, so betrachtet die Shell dieses als ein sogenanntes *pattern*¹, welches eine Vielzahl von Dateinamen abdecken kann.

Jedes in der Kommandozeile gefundene *pattern* wird dann von der Shell expandiert, d.h. durch alle Dateinamen ersetzt², die es abdeckt. Falls kein Dateiname gefunden werden kann, den ein vorgegebenes *pattern* abdeckt, wird das entsprechende *pattern* nicht expandiert und unverändert dem aufgerufenen Kommando oder Shell-Skript übergeben.

Zur Expandierung von Dateinamen stehen nun folgende Metazeichen zur Verfügung:

Metazeichen	Bedeutung
*	steht für »eine beliebige Zeichenfolge« (auch die leere) ab* deckt alle Dateinamen ab, die mit <i>ab</i> beginnen und dann beliebige weitere Zeichen oder auch kein weiteres Zeichen enthalten: <i>ab</i> , <i>abc</i> , <i>aber008.c</i> , usw. x*.c deckt alle Dateinamen ab, die mit <i>x</i> beginnen und mit <i>.c</i> enden: <i>x.c</i> , <i>xyz.c</i> , <i>xalt.c</i> , <i>xvers145.c</i> , usw.
?	steht für »ein beliebiges einzelnes Zeichen« ab? deckt alle Dateinamen ab, die mit <i>ab</i> beginnen und dann genau ein weiteres Zeichen enthalten: <i>abc</i> , <i>abx</i> , <i>ab8</i> usw. (nicht abgedeckt würden Namen wie <i>ab</i> , <i>abcd</i> , <i>ab82</i> , <i>abxx</i> , usw.) add?.c deckt alle Dateinamen ab, die mit <i>add</i> beginnen, dann ein beliebiges weiteres Zeichen haben und mit <i>.c</i> enden: <i>add1.c</i> , <i>add9.c</i> , <i>addb.c</i> usw. (nicht abgedeckt würden Namen wie <i>add10.c</i> , <i>add.c</i> , <i>addiere.c</i> , usw.)
[...]	steht für »eines der in [...] angegebenen Zeichen« *.[ch] deckt alle Dateinamen ab, die mit <i>.c</i> oder <i>.h</i> enden: <i>add.c</i> , <i>stdio.h</i> , <i>sub2div.h</i> usw. obst[1234] deckt die Dateinamen <i>obst1</i> , <i>obst2</i> , <i>obst3</i> und <i>obst4</i> ab. Bei der Angabe der Zeichen innerhalb von [...] sind auch Bereichsangaben wie <i>[a-f]</i> oder <i>[0-8]</i> oder <i>[A-Z]</i> erlaubt.

Tabelle 4.4: Metazeichen für Dateinamenexpandierung

1. Deutsch: Mustervorgabe, Schablone.
2. Alphabetisch sortiert.

Metazeichen	Bedeutung
[!...]	obst[1-4a] deckt die Dateinamen <i>obst1</i>, <i>obst2</i>, <i>obst3</i>, <i>obst4</i> und <i>obsta</i> ab. steht für »ein Zeichen, welches nicht in [!...] angegeben ist« *.[!ch] deckt alle Dateinamen ab, die als vorletztes Zeichen einen Punkt besitzen, aber nicht mit <i>c</i> oder <i>h</i> enden: <i>add.p</i>, <i>obst4</i>, <i>add2.2</i> usw. Bei der Angabe der Zeichen innerhalb von [!...] sind auch Bereichsangaben wie [!a-z] oder [!3-7] oder [!A-L] erlaubt. *[!1-9] deckt alle Dateinamen ab, die nicht mit einer Ziffer enden.

Tabelle 4.4: Metazeichen für Dateinamenexpandierung (Fortsetzung)

Ausnahmen zu obigen Regeln:

Folgende Zeichenfolgen in Dateinamen werden nur dann abgedeckt, wenn sie explizit im entsprechenden pattern angegeben wurden:

. (Punkt) am Anfang eines Dateinamens
/.
/

Beispiel

```
rm *
```

Alle Dateien des Working-Directorys (außer die, deren Name mit . beginnt) löschen.

```
ls *.c
```

Alle Dateien auflisten, deren Name mit .c endet.

```
cp /usr/bin/k? /user1/emil
```

Alle Dateien aus dem Directory */usr/bin*, deren Name mit *k* beginnt und insgesamt nur zwei Zeichen lang ist, in das Directory */user1/emil* kopieren.

```
ls [a-di-ly]*
```

Alle Dateien auflisten, deren Name mit Buchstaben *a*, *b*, *c*, *d*, *i*, *j*, *k*, *l* oder *y* beginnt.

```
cp [!a-zA-Z0-9]* /tmp
```

Alle Dateien, deren Name nicht mit einem Buchstaben oder einer Ziffer beginnt, in das Directory */tmp* kopieren.

Zunächst werden zu Übungszwecken einige leere Dateien (mit dem Kommando **touch**) angelegt, bevor dann hierfür Beispiele der Dateinamenexpandierung gezeigt werden:

```
$ touch .fritz
$ touch franz
$ touch fratz
$ touch fritz
$ touch frutz
$ touch .katz
$ ls f*z
franz
fratz
fritz
frutz
$ ls fr?tz
fratz
fritz
frutz
$ ls fr[a-i][nt]z
franz
fratz
fritz
$ ls fr[!a-i]tz
frutz
$ ls -CF .*
.fritz .katz

.:
adress*      countdown*  eing_zaehl  kinder1*    weiblich*
adresse.txt  cph*        franz        kinder2*    which1*
anfang*      datum*      fratz        ll*          woist*
argu_pid*    demo*       fritz        lszaehl     zaehle.txt
ausgab*      div_null*   frutz        schiebe*    zeig_pid*
core         div_null.c  gls*         stdio.h
countdow2*   div_null2*  gruss*       var_wert*

...:
shellueb/    uebung1/    uebung3/
....
....
$ ls $HOME/.[a-z]*
/user1/egon/.exrc  [Mögliche Ausgabe]
/user1/egon/.mailrc
/user1/egon/.news_time
/user1/egon/.profile
$ echo fr[xyz][!abc]*
fr[xyz][!abc]*    [Da pattern keinen Dateinamen abdeckt, keine
Dateinamenexp.]

$ echo *
adress adresse.txt anfang argu_pid ausgab core countdow2 countdown cph datum
demo div_null div_null.c div_null2 eing_zaehl franz fratz fritz frutz gls
gruss kinder1 kinder2 ll lszaehl schiebe stdio.h var_wert weiblich which1 wo-
ist zaehle.txt zeig_pid
$ echo .*
```

```
. .. .fritz .katz
$ echo */*
*/*
[kein Subdirectory im Working-Directory]
$ rm -i .*
rm: . directory
rm: .. directory
.fritz ? y
.katz ? y
$ rm -i fr*
franz ? y
fratz ? y
fritz ? y
frutz ? y
$
```

Hinweis

Die Dateinamenexpansion kann auch ausgeschaltet werden, indem entweder beim Shell-Aufruf die Option **-f** angegeben oder aber diese Option mit dem builtin-Kommando **set** eingeschaltet wird:

```
$ echo d*
datum demo div_null div_null.c div_null2
$ set -f
$ echo d*
d*
$ set +f
$
```

4.8 Quoting

Die in Kapitel 4.1 vorgestellten Metazeichen haben eine Sonderbedeutung für die Shell. In manchen Anwendungsfällen ist es nun notwendig, die Sonderbedeutung dieser Zeichen kurzfristig auszuschalten.

Dazu steht der Quoting-Mechanismus der Shell zur Verfügung.

4.8.1 Verschiedene Arten von Quoting

Mit Quoting können – wie schon erwähnt – die Sonderbedeutungen der Metazeichen ausgeschaltet und diese Zeichen als »ganz normale Zeichen« verwendet werden. Es existieren verschiedene Möglichkeiten des Quotings:

1. Voranstellen von \
2. Klammerung mit '..'
3. Klammerung mit "..."

Voranstellen von \

Wird einem der Metazeichen ein \ vorangestellt, so verliert dieses Metazeichen seine Sonderbedeutung.

Beispiel

Zunächst werden wieder zu Übungszwecken leere Dateien angelegt, bevor dann der Quoting-Mechanismus an diesen Dateinamen vorgestellt wird:

```
$ touch dem? ↵
$ touch film* ↵
$ touch filmapparat ↵
$ touch filmkosten ↵
$ touch filmstar ↵
$ ls de* ↵
dem?
demo
$ ls dem? ↵
dem?
demo
$ ls dem\? ↵
dem?
$ rm dem\? ↵      [es wird nur dem? (nicht demo) gelöscht]
$ ls de* ↵
dem?
$ ls film* ↵
film*
filmapparat
filmkosten
filmstar
$ ls film\* ↵
film*
$ rm film\* ↵      [nur film* wird gelöscht]
$ ls film* ↵
filmapparat
filmkosten
filmstar
$
```

Die Angabe von \ unmittelbar vor dem \$-Zeichen schaltet dessen Sonderbedeutung (Zugriff auf den Wert einer Variablen) aus, was dazu führt, daß \$ als normales Zeichen verwendet wird:

```
$ echo \$HOME=$HOME ↵
$HOME=/user1/egon
$
```

Normalerweise würde das Neuezeile-Zeichen, das von der Taste ↵ erzeugt wird, das Ende der Kommandozeile anzeigen und die Shell zur Abarbeitung der angegebenen Kommandozeile veranlassen. Die unmittelbare Eingabe von \ vor dem Drücken der Taste ↵ bewirkt, daß diese Sonderbedeutung des Neuezeile-

Zeichens ausgeschaltet wird. Daß eine Kommandozeile noch nicht abgeschlossen ist, zeigt die Shell durch die Ausgabe des Sekundär-Promptstrings an.

Nachdem ein nicht ausgeschaltetes Neuezeile-Zeichen (Taste ) eingegeben wurde, werden alle Paare \Neuezeile-Zeichen von der Shell vollständig aus der Kommandozeile entfernt, bevor die Shell Kommando- und Parametersubstitution durchführt:

```
$ echo $H0\   
> ME   
/user1/egon  
$
```

Wenn sich also eine Kommandozeile über mehr als eine Zeile erstrecken soll, so kann \ als Fortsetzungszeichen verwendet werden.

Klammerung mit '..'

Alle Metazeichen zwischen zwei einzelnen Apostrophen¹ (außer einem weiteren Apostroph) verlieren ihre Sonderbedeutung.

Beispiel

```
$ cat >dem\?   
Dies ist Text in dem?   
[Strg-D]  
$ touch film\*   
$ cat dem?   
Dies ist Text in dem? [Ausgabe des Inhalts der Dateien dem? und demo]  
echo "Gib eine Zeile von Text ein"  
read x  
echo $x  
echo "Gib noch eine Textzeile ein"  
read x y z  
echo $x  
echo $y  
echo $z  
$ cat 'dem?'   
Dies ist Text in dem? [Nur Ausg. des Inhalts der Datei dem? (nicht von demo)]  
  
$ cat dem'?'   
Dies ist Text in dem? [Nur Ausg. des Inhalts der Datei dem? (nicht von demo)]  
  
$ ls film*   
film*  
filmapparat  
filmkosten  
filmstar  
$ ls film'*'   
film*
```

1. Nicht zu verwechseln mit den Gegen-Apostrophen der Kommandosubstitution `..`


```
$ rm film*' [↵] [nur film* wird gelöscht]
$ ls film* [↵]
filmapparat
filmkosten
filmstar
$ echo '$HOME=$HOME' [↵] [Sonderbedeutung von $ einmal ausgeschaltet]
$HOME=/user1/egon
$ echo '$$HOME=$HOME' [↵] [Sonderbedeutung von $ ebenfalls einmal ausgeschaltet]
$HOME=/user1/egon
$
```

Innerhalb von `'..'` verliert sogar das zuvor vorgestellte Quoting-Zeichen `\` seine Sonderbedeutung. Das nachfolgende Beispiel zeigt, daß diese Art des Quotings beim Vorkommen vieler auszuschaltender Metazeichen in einer Kommandozeile dem Quoting mit `\` vorzuziehen ist, da sich in diesem Fall doch wesentlich lesbarere Kommandozeilen ergeben:

```
$ echo 'Wasser & Feuer ($ \ <DM>)' [↵]
Wasser & Feuer ($ \ <DM>)
$ echo Wasser \& Feuer \(\$ \ \ \<DM\>\) [↵]
Wasser & Feuer ($ \ <DM>)
$
```

Zwar kann auch mit dieser Quoting-Art die Sonderbedeutung des Neuezeilen-Zeichens (Abschluß einer Kommandozeile) ausgeschaltet werden; allerdings wird anders als bei `\` [↵] der dadurch erzeugte Zeilenvorschub nicht von der Shell entfernt.

```
$ echo $H0' [↵]
> 'ME' [↵]
ME [Ausgabe von $H0: leere Zeichenkette]
$ [Ausgabe des Strings ME in einer neuen Zeile]
```

Wenn sich eine Kommandozeile über mehr als eine Zeile erstrecken soll, so wird meist `\` als Fortsetzungszeichen verwendet; Klammerung mit `'..'` wird meist dann verwendet, wenn ein Text, der über mehrere Zeilen auszugeben ist, mit einem Kommando angegeben wird.

Beispiel

```
$ ls -F | grep *$ ; \ [↵]
> echo '-----' [↵]
> Dies sind alle ausfuehrbaren Dateien im Directory: [↵]
> ' $HOME/shellueb' [↵]
adress*
...
...
zeig_pid*
-----
Dies sind alle ausfuehrbaren Dateien im Directory:
```

```
/user1/egon/shellueb  
$
```

Um die Sonderbedeutung eines einfachen Apostrophs innerhalb einer mit `'..'` geklammerten Zeichenkette auszuschalten, muß dieser mit Anführungszeichen geklammert werden: `'..""..'`

Beispiel

```
$ finde_c='find . -name ""*.c"" -print'  
$ echo $finde_c >finde_c  
$ cat finde_c  
find . -name "*.c" -print  
$ chmod u+x finde_c  
$ finde_c  
./div_null.c  
$
```

Klammerung mit `".."`

Bei einer Klammerung mit `".."` verlieren die meisten, aber nicht alle Metazeichen ihre besondere Bedeutung: Innerhalb von `".."` behalten nur die Metazeichen

`\ " ' $`

ihre Sonderbedeutung.

Im Unterschied zur Apostroph-Klammerung schaltet diese Form der Klammerung also folgendes nicht aus:

- ▶ das Quoting mit `\`
- ▶ Parametersubstitution (`$variable`: Zugriff auf den Wert von *variable*)
- ▶ Kommandosubstitution (``kdo``)

Da auch `\` innerhalb von `".."` seine Sonderbedeutung behält, kann es verwendet werden, um die Sonderbedeutung der 4 Zeichen `\ " ` $` innerhalb von `".."` auszuschalten.

Beispiel

```
$ cat muenz_kom  
muenze=zahl  
echo 1: '$muenze'  
echo 2: "$muenze"  
echo 3: "\$muenze"  
echo "Die amerikanische Waehrung ist: \$( 'Dollar')." "  
echo "\`pwd\` gibt Working-Directory aus, welches ist: `pwd`"  
$ chmod +x muenz_kom  
$ muenz_kom  
1: $muenze  
2: zahl  
3: $muenze  
Die amerikanische Waehrung ist: $ ( 'Dollar').  
`pwd` gibt Working-Directory aus, welches ist: /user1/egon/shellueb
```

```
$ touch film\*  
$ ls film"*"  
film*  
$ rm film"*" [nur film* wird gelöscht]  
$ ls film*  
filmapparat  
filmkosten  
filmstar  
$ rm film* [löscht alle Dateien, deren Name mit film beginnt]  
$ ls dem"?"  
dem?  
$ rm dem"? " [nur dem? (nicht Datei demo) wird gelöscht]  
$
```

Ähnlich wie bei `..` wird auch bei dieser Quoting-Art die Sonderbedeutung des Neuezeile-Zeichens (Abschluß einer Kommandozeile) zwar ausgeschaltet, der dadurch erzeugte Zeilenvorschub aber nicht von der Shell entfernt.

```
$ echo $H0"  
> "ME"  
[Ausgabe von $H0: leere Zeichenkette]  
ME [Ausgabe des Strings ME in einer neuen Zeile]  
$
```

Somit kann auch diese Quoting-Art verwendet werden, wenn ein Text, der sich über mehrere Zeilen erstreckt, mit einem Kommando auszugeben ist. Im Unterschied zu `..` können allerdings hier Parameter- und Kommandosubstitution innerhalb von `".."` durchgeführt werden:

Beispiel

```
$ ls -F | grep *$ ; \  
> echo "-----"  
> Dies sind alle ausfuehrbaren Dateien im Directory:  
> `pwd`"  
adress*  
...  
...  
zeig_pid*  
-----  
Dies sind alle ausfuehrbaren Dateien im Directory:  
/user1/egon/shellueb  
$ echo "Menue:"  
> Kopieren : k  
> Verlagern : v  
> Umbenennen : u  
> Verlassen : q  
> Gib entsprechendes Zeichen ein:"
```

```
Menue:  
Kopieren : k  
Verlagern : v
```

```
Umbenennen : u
Verlassen : q
Gib entsprechendes Zeichen ein:
$
```

Obwohl die Angaben **\$#** und **\$@** die ganze Liste der Positionsparameter (**\$1**, **\$2**, ...) repräsentieren, werden sie doch unterschiedlich interpretiert, wenn sie innerhalb von **".."** angegeben sind:

"\$*" entspricht "\$1 \$2 \$3"

Die ganze Parameterliste wird als ein einziges Wort interpretiert, wobei die einzelnen Parameterwerte mit Leerzeichen voneinander getrennt sind.

"\$@" entspricht "\$1" "\$2" ...

Die Liste aller Parameter wird hierbei als Aneinanderreihung von einzelnen Wörtern interpretiert.

Diese Unterscheidung ist vor allen Dingen innerhalb von Shell-Skripts bei der Auswertung von übergebenen Argumenten wichtig:

Beispiel

```
$ cat argaus ↵
echo "Die Parameterzahl ist $#"
```

```
echo "\$1 ist ($1)"
echo "\$2 ist ($2)"
$ chmod u+x argaus ↵
$ set Egon Miller ↵
$ argaus "$*" ↵
Die Parameterzahl ist 1
$1 ist (Egon Miller)
$2 ist ()
$ argaus "$@" ↵1
Die Parameterzahl ist 2
$1 ist (Egon)
$2 ist (Miller)
$
```

Kommt **\Neuzeile-Zeichen** innerhalb von **".."** vor, wird dieses Zeichenpaar zuerst entfernt, bevor Parameter- und Kommandosubstitution durchgeführt werden:

```
$ curdir="`p\` ↵
> w\ ↵
> d\ ↵
> "` ↵
$ echo $curdir ↵
/user1/egon/shellueb
$
```

1. @ ist eventuell mit \@ einzugeben, um seine Funktion "Letztes Zeichen löschen" auszuschalten.

Wird ein `\` innerhalb von `".."` vor einem anderen Zeichen als `\` ``` `"` `$` oder Neuezeile-Zeichen angegeben, so repräsentiert es sich selbst.

Beispiel `$ echo "Das \-Zeichen"`
 Das \-Zeichen
 \$

Oft ist es notwendig, Kommandosubstitutionen mit `".."` zu klammern, um fakultativ bei den dadurch bereitgestellten Zeichenketten die darin enthaltenen Metazeichen (wie z.B. Dateinamenexpansion) auszuschalten.

Beispiel `$ cat abc`
 echo Ausfuehrbare Dateien, die mit a, b oder c beginnen
 echo `ls -F [abc]\* | grep \*\$`
 \$ `chmod u+x abc`
 \$ `abc`
 Ausfuehrbare Dateien, die mit a, b oder c beginnen
 abc adress adresse.txt anfang argaus argu_pid ausgab
 countdow2 countdown cph [Falsche Ausgabe: adresse.txt nicht ausfuehrbar¹]
 \$ `cat abc2`
 echo Ausfuehrbare Dateien, die mit a, b oder c beginnen
 echo "`ls -F [abc]\* | grep \*\$`" [Kommandosubst. mit `".."` geklammert]
 \$ `chmod u+x abc2`
 \$ `abc2`
 Ausfuehrbare Dateien, die mit a, b oder c beginnen
 abc*
 abc2*
 adress* [Nun ist die Ausgabe richtig]
 anfang* [die von Kommandosubst. gelieferten Zeilenvorschübe]
 argaus* [bleiben erhalten]
 argu_pid*
 ausgab*
 countdow2*
 countdown*
 cph*
 \$

Hinweis Die Erkennung von builtin-Kommandos² durch die Shell kann niemals durch Quoting unterbunden werden:

```
$ \p\w\ld
/user1/egon/shellueb
$ "pwd"
/user1/egon/shellueb
$ 'pwd'
```

1. Das von der Kommandosubstitution gelieferte Wort `adress*` wird bei der Ausführung des `echo`-Kommandos expandiert.
2. In Kapitel 4.15.8 sind alle builtin-Kommandos zusammengefaßt.

```

/user1/egon/shellueb
$ "cd" /bin
$ 'pwd'
/bin
$ \c\d
$ \c'd' shellueb
$ \p"wd"
/user1/egon/shellueb
$ "cd /bin"
cd /bin: not found          ["cd bin" wird als Kdoname interpretiert
                             (existiert nicht)]
$

```

4.8.2 Quoting innerhalb von Kommandosubstitution

Das Zeichen \ kann innerhalb von Kommandosubstitution verwendet werden, um die Sonderbedeutung von \ bzw. ` (*backquote*) auszuschalten.

Da die Shell beim jeweiligen Lesen der Kommandozeile voranstehende \ entfernt, kann hieraus eine erneute Kommandosubstitution in der Kommandozeile entstehen, welche die Shell dazu zwingt, die entstandene Kommandozeile erneut zu lesen. Hierbei werden voranstehende \ entfernt, was wieder zu neuen Kommandosubstitutionen führen kann, usw.

Beispiel

Eine Datei *u* enthalte den Dateinamen *uv*. Die Datei *uv* wiederum enthalte den Dateinamen *uvw*. Die Datei *uvw* enthalte ihrerseits den Dateinamen *uvw**x*. Die Datei *uvw**x* schließlich enthalte die Zeile

"Stille Wasser, die sind tief":

```

——u——
|uv  ——uv——
      |uvw  ——uvw——
            |uvwx ——uvwx——
                               |Stille Wasser, die sind tief

```

Der Aufruf

```
cat `cat \cat \\\cat u\\\`
```

würde dann den Inhalt der Datei *uvw**x* ausgeben:

Stille Wasser, die sind tief

```

cat   `cat`  \`cat`  \\`cat`  u\\`\\`
      |
      |
      v
`cat  \`cat`  \\`cat`  u\\`\\`    ----> cat  uvwx
      |                                     ^
      |1.Kommandosubst.                  |
      v                                   uvwx
\`cat  \`cat`  u\\`                ----> \`cat uvw`
      |                                     ^
      |2.Kommandosubst.                  uvw
      v                                   |
\underline{\`cat}  \underbrace{\`cat}_{} u\\`    ----> \underbrace{\`cat}_{}`uv`
      |                                     ^
      |3.Kommandosubst.                  \underbrace{\`cat}_{}`uv`
      v                                   |
\underline{\`cat}  \underline{u}\\`            ----> \underline{uv}
      |
      |3.Kommandosubst. (liefert uv)
      v
      uv

```

Beispiel

```
[\ entfernt, da in ".." eingebettet, so]
[daß Kommandosubst. Hallo Franz und
nicht Hallo "Franz" liefert]
```

Beispiel

```
[echo gibt dann nur Leerzeile aus]
```

\$

Backslashes, die in einer Kommandosubstitution vor einem anderem Zeichen als `\`"$` oder *Neuzeile-Zeichen* angegeben sind, werden beim Lesen des Kommandos in der Kommandosubstitution ``..`` nicht entfernt.

Beispiel

```
$ touch abc*  
$ echo "abc-Dateien:\n`ls abc*`" # unter Linux (bash): echo -e "..."  
abc-Dateien:  
abc  
abc*  
abc2  
$ echo "abc-Dateien:\n`ls abc\*`" # unter Linux (bash): echo -e "..."  
abc-Dateien:  
abc*  
$
```

Es ist weiter möglich, Kommandosubstitution für Kommandos durchzuführen, die in einer Datei angegeben sind.

Beispiel

```
$ cat pd  
pwd  
$ chmod u+x pd  
$ echo ``cat pd``  
/user1/egon/shellueb  
$
```

Da in einer Kommandosubstitution (beim Lesen) ein `\` vor einem `$` entfernt wird, muß die Sonderbedeutung eines `$` mit `\\$` oder `\\\$(` (nicht mit `\$`) ausgeschaltet werden.

Beispiel

```
$ name=franz  
$ echo `echo \${name}`  
franz  
$ echo `echo \\${name}`  
$name  
$ echo `echo \\\${name}`  
$name  
$ echo ``echo \${name}``  
franz  
$
```


Kurze Zusammenfassung:

Die folgende Tabelle gibt eine Übersicht über die Gültigkeit bestimmter Metazeichen bei den unterschiedlichen Quoting-Arten und der Kommandosubstitution:

	Metazeichen						
Quoting	\	\$	*?[	`	"	'	
\	–	–	–	–	–	–	–
".."	x	x	–	x	+	–	v
'..'	–	–	–	–	–	+	v
`..`	x	x	x	+	x	x	x

Hierbei bedeutet:

- Sonderbedeutung ausgeschaltet
- x behält seine Sonderbedeutung
- + beendet entspr. Quoting bzw. Kommandosubstitution
- v Sonderbedeutung (Kommandoabschluß) ausgeschaltet, aber Bedeutung »Zeilenvorschub« bleibt erhalten

4.9 Ein- und Ausgabeumlenkung

Wie bereits im ersten Buch besprochen, lesen sehr viele UNIX-Kommandos von der Standardeingabe und schreiben auf die Standardausgabe bzw. Standardfehlerausgabe.

Die üblichen Voreinstellungen sind dabei:

Standardeingabe: Dialogstation (Tastatur)
 Standardausgabe: Dialogstation (Bildschirm)
 Standardfehlerausgabe: Dialogstation (Bildschirm)

Zwischen Standardausgabe und Standardfehlerausgabe wird unterschieden, um echte Daten (Standardausgabe) und Fehler- oder Diagnosemeldungen (Standardfehlerausgabe) trennen zu können.

Somit ergibt sich Abbildung 4.13 für viele UNIX-Kommandos:

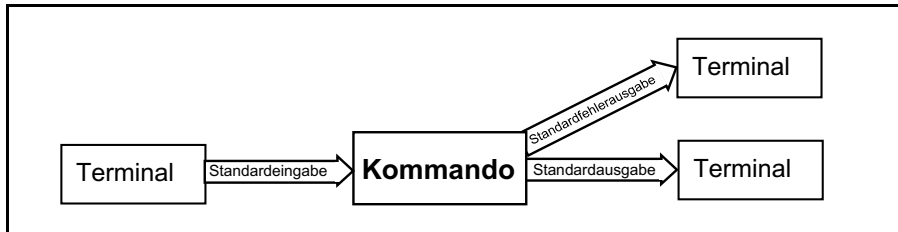


Abbildung 4.13: Standardeingabe, Standardausgabe und Standardfehlerausgabe

Jedem dieser drei Ein-/Ausgabekanäle ist ein sogenannter Dateideskriptor¹ zugeordnet:

Standardeingabe (**stdin**): 0
Standardausgabe (**stdout**): 1
Standardfehlerausgabe (**stderr**): 2

Anderen innerhalb eines Programms oder Kommandos explizit eröffneten Dateien werden die Dateideskriptoren 3, 4, 5, 6 usw. zugeordnet.

Diese Ein-/Ausgabekanäle können in Dateien umgelenkt werden. Eine solche Umlenkung wird durch die Angabe von speziellen Zeichen erreicht:

<datei

lenkt die Standardeingabe (von der Dialogstation) in die Datei *datei* um, d.h., es wird nicht mehr von der Dialogstation, sondern aus der Datei *datei* gelesen.

Beispiel mail emil <glueckwunsch

Hier wird mail an **emil** geschickt, wobei der Inhalt des zu sendenden Briefes nicht von der Tastatur, sondern von der Datei *glueckwunsch* gelesen wird.

Obwohl jeder der beiden Aufrufe

sort namdatei (1) und
sort < namdatei (2)

zur sortierten Ausgabe der Datei *namdatei* führt, besteht doch ein Unterschied zwischen diesen beiden:

Während der Ausführung von (1) wird die Standardeingabe nicht umgelenkt, weshalb sie weiter für Eingaben über die Tastatur zur Verfügung stehen würde. In diesem Fall öffnet das Kommando **sort** selbst die angegebene Datei *namdatei* und liest aus ihr die zu sortierenden Daten. Bei (2) dagegen gilt dies nicht, da die Standardeingabe vor der Ausführung von **sort** von der Shell in die Datei *namda-*

1. Erinnerung: In UNIX werden Geräte wie Dateien behandelt.

tei umgelenkt wird, so daß **sort** nicht direkt, sondern über die umgelenkte Standardeingabe aus *namdatei* liest. Bei (2) wäre also keine Eingabe über die Tastatur möglich, wenn **sort** solche Eingaben verlangen würde.

Deutlicher wird es vielleicht bei den folgenden Aufrufen, wobei die Angabe – auf der Kommandozeile **sort** mitteilt, daß es hier von der Standardeingabe (bis *Strg-D*) lesen soll:

```
$ sort uvwx - 
Morgenstund       [Eingabe zusätzlich zu sortierender Daten von Tastatur]
hat 
Gold 
im 
Mund 
Strg-D
Gold
Morgenstund
Mund
Stille Wasser, die sind tief [Inh. von uvwx; in eingeg. Zeilen einsortiert]
hat
im
$
```

Würde Eingabeumlenkung bei diesem Aufruf verwendet, so bestünde keine Möglichkeit, Daten über Tastatur – zum Sortieren – mit einzugeben, da in diesem Fall die Standardeingabe bereits in die Datei *uvwx* umgelenkt wäre:

```
$ sort <uvwx - 
Stille Wasser, die sind tief      [Daten-Eingabe über Tastatur nicht möglich]
$
```

>datei

lenkt die Standardausgabe (von der Dialogstation) in die Datei *datei* um, d.h., es wird nicht mehr auf die Dialogstation, sondern in die Datei *datei* geschrieben.

Beispiel

```
$ cat namdatei 
hans
fritz
anton
emil
$ sort namdatei 
anton
emil
fritz
hans
$ sort namdatei > sortiert 
$ cat sortiert 
anton
```

```

emil
fritz
hans
$ ls -CF . > wd_inhalt
$ cat wd_inhalt
abc*      core      eing_zaehl  namdatei   var_wert*
abc*      countdow2* finde_c*    pd*        wd_inhalt
abc2*     countdown* gls*        schiebe*   weiblich*
adress*   cph*        gruss*      sortiert   which1*
adresse.txt datum*      kinder1*    stdio.h    woist*
anfang*   demo*       kinder2*    u          zaehle.txt
argaus*   div_null*   ll*         uv         zeig_pid*
argu_pid* div_null.c  lszaehl     uvw
ausgab*   div_null2* muenz_kom* uvwx
$

```

Hinweis

Wenn die Datei *datei* bereits existiert, so wird ihr alter Inhalt überschrieben. Existiert *datei* noch nicht, so wird sie neu angelegt:

```

$ echo `pwd` >wd_inhalt
$ cat wd_inhalt
/user1/egon/shellueb
$

```

Das mit dem Kommando

```
cp div_null.c /tmp/div_null.c
```

bewirkte Kopieren der Datei *div_null.c* könnte auch mit einer der beiden folgenden Kommandozeilen erreicht werden:

```
cat div_null.c >/tmp/div_null.c oder
cat <div_null.c >/tmp/div_null.c
```

Die Eingabe *>datei* (ohne Angabe eines Kommandos) erzeugt eine leere Datei mit Namen *datei* und lenkt nicht die Standardausgabe der momentan aktiven Shell in die Datei *datei* um. Um ein Umlenken der Standardausgabe für die momentan aktive Shell zu erreichen, wäre das builtin-Kommando **exec**¹ verwendet worden.

Beispiel

```

$ pwd
/user1/egon/shellueb
$ >neudat [Anlegen einer leeren Datei mit Namen neodat]
$ ls n*
namdatei
neudat
$ exec >logfile [Standardausgabe der Shell in logfile umlenken]
$ echo "Working-Directory: `pwd`"
$ ls -CF a*

```

1. Eine genaue Beschreibung dieses Kommandos wird in Kapitel 4.15.3 gegeben.

```
$ exec >/dev/tty [Standardausgabe wieder auf Bildschirm lenken]
$ cat logfile
Working-Directory: /user1/egon/shellueb
abc*      abc2*      adresse.txt  argaus*      ausgab*
abc*      adress*     anfang*     argu_pid*
```

>>datei

lenkt ebenfalls die Standardausgabe (von der Dialogstation) in die Datei *datei* um; allerdings wird hierbei der alte Inhalt einer eventuell schon existierenden Datei *datei* nicht überschrieben. Die neuen Ausgabedaten werden an das Ende von *datei* geschrieben. Sollte die Datei *datei* noch nicht existieren, wird sie wie bei der Konstruktion *>datei* neu angelegt.

Beispiel

```
$ echo "--- `date` ---" >mom_inhalt
$ ls -C >> mom_inhalt
$ cat mom_inhalt
--- Thu Nov 14 14:53:43 GMT 1996 ---
abc      ausgab      div_null.c  ll      schiebe      wd_inhalt
abc*     core          div_null2   logfile  sortiert     weiblich
abc2     countdow2     eing_zaehl  lszaehl  stdio.h      which1
adress   countdown     finde_c     mom_inhalt  u          woist
adresse.txt  cph          gls          muenz_kom  uv         zaehle.txt
anfang   datum         gruss       namdatei   uvw        zeig_pid
argaus   demo          kinder1     neudat     uvwx
argu_pid div_null      kinder2     pd         var_wert
```

Zuerst wird das heutige Datum in die Datei *mom_inhalt* geschrieben, wobei ein eventueller Inhalt dieser Datei überschrieben wird. Danach werden alle Dateinamen des Working-Directorys ans Ende der Datei *mom_inhalt* geschrieben.

<<wort

Hier-Dokument (engl. *here document*): Nachdem die Parameter- und Kommandosubstitution für *wort* durchgeführt ist, wird die Eingabe an die Shell Zeile für Zeile gelesen, bis eine Zeile gefunden wird, die genau mit *wort* übereinstimmt, oder bis ein **EOF** gelesen wird.

Beispiel

```
$ sort <<ENDE
> Hans
> Fritz
> Anni
> ENDEN
> ENDE
Anni
ENDEN
Fritz
Hans
```

```
$ cat <<ENDTEXT↵
> Menu:↵
>   Neu eingeben:  n↵
>   Aendern:      a↵
>   Loeschen:     l↵
>   Drucken:      d↵
>   Ende:         e↵
> ENDTEXT↵
Menue:
    Neu eingeben:  n
    Aendern:      a
    Loeschen:     l
    Drucken:      d
    Ende:         e

$
```

Das Eingabeende wird abhängig davon, ob Quoting im *wort* verwendet wurde oder nicht, unterschiedlich festgelegt:

Quoting im *wort*

In den folgenden Eingabezeilen wird die Sonderbedeutung der Metazeichen ausgeschaltet, d.h., daß die eingegebenen Zeichen ohne besondere Interpretation von der Shell gelesen werden.

Beispiel

```
$ cat <<\$HOME↵
> `pwd`↵
> echo $HOME↵
> \$HOME↵
> \`pwd\`↵
> $HOME↵
`pwd`
echo $HOME
\$HOME
\`pwd\`
$
```

Kein Quoting im *wort*

Für die folgenden Eingabezeilen gelten dann folgende Regeln:

1. Parameter- und Kommandosubstitution findet statt.
2. *\Neuzeile-Zeichen-Kombinationen* werden ignoriert.
3. ** muß verwendet werden, um die Sonderbedeutung der Zeichen ** sowie *\$* und *`* auszuschalten.

Beispiel

```
$ echo $HOME
/user1/egon
$ cat <<$HOME
> `logname`
> echo $HOME
> \ $HOME
> ``logname``
> /user1/egon
egon
echo /user1/egon
$HOME
`logname`
$
```

Typische Anwendungen für das Hier-Dokument sind:

- ▶ Erstellen und Kompilieren eines C-Programms in einem Shell-Skript
- ▶ Angabe von ed-Skripts zum Editieren von Texten aus Dateien, Variablen oder von Eingabezeilen
- ▶ Senden von mail mit einem im Skript vorgegebenen Text

Beispiel

```
$ cat hallo.arc
cat << --ENDE-- > hallo.c
main()
{
    printf("Hallo egon\n");
}
--ENDE--
cc -o hallo hallo.c
echo "hallo.c kompiliert
Programmname: hallo"
$ chmod u+x hallo.arc
$ hallo.arc
hallo.c kompiliert
Programmname: hallo
$ hallo
Hallo egon
$ cat which2
tmp_name=/tmp/${LOGNAME}.$$
echo $PATH >$tmp_name

# Ersetzen von leeren Pfaden durch . (Working-Directory)
# Es muss fuer alle drei Moeglichkeiten:
#   - leerer Pfad am Anfang (^:); ^ steht fuer Zeilenanfang
#   - leerer Pfad in der Mitte (::)
#   - leerer Pfad am Ende (:$); $ steht fuer Zeilenende
# ein eigenes ed-Skript angegeben werden, da ed sich beim Lesen
# von Kommandos aus einem ed-Skript beim Vorkommen des ersten
# Fehlers automatisch beendet
ed -s $tmp_name <<ende_ed >/dev/null # am Anfang
```

```

s/^:./:/
w
ende_ed
ed -s $tmp_name <<ende_ed >/dev/null # in der Mitte
s/::./:/
w
ende_ed
ed -s $tmp_name <<ende_ed >/dev/null # am Ende
s/:$:./:/
w
ende_ed

pfad="`cat $tmp_name | tr : ' '`"
find $pfad -name $1 -print
rm $tmp_name
$ chmod u+x which2
$ echo $PATH
:/bin:/usr/bin
$ which2 dircmp
/usr/bin/dircmp
$ which2 du
/bin/du
$ which2 which2
./which2
$

```

Im vorhergehenden Skript *which2* wurde folgende Zeile angegeben:

```
tmp_name=/tmp/${LOGNAME}.$$
```

Dabei liefert \$\$ die PID der jeweiligen Subshell, die das Skript ausführt. Solche Konstruktionen werden häufig verwendet, um eindeutige Namen für temporäre Dateien zu erhalten, da es unwahrscheinlich ist, daß ein anderer Benutzer in */tmp* ebenfalls einen Dateinamen wählt, der mit genau dieser angehängten Prozeßnummer (PID) endet.

Das folgende Skript *diskuse* gibt die Speicherplatz-Belegung für das ganze Dateisystem / in die Datei */user1/egon/.diskuse* aus. Am Ende sendet es mail an **egon**, indem es ihm dies mitteilt:

```

$ cat diskuse
du / >$HOME/.diskuse
mail $LOGNAME <<ENDE_NACHRICHT
    Die Belegungsstatistik fuer das Dateisystem /
    befindet sich in der Datei $HOME/.diskuse
ENDE_NACHRICHT
$ chmod u+x diskuse
$

```


<<-wort

entspricht weitgehend der Angabe `<<wort`, allerdings werden hier in den nachfolgenden Eingabezeilen alle führenden Tabulatorzeichen ignoriert. Dies ist besonders bei Shell-Skripts von Nutzen, in welchen oft Tabulatorzeichen zum Einrücken von Text verwendet werden, um die Lesbarkeit zu erhöhen.

<& fd

verwendet die Datei, welche mit dem Dateideskriptor `fd` verbunden ist, als Standardeingabe.

Beispiel

```
mail hans < geburtstag
```

ist identisch zu

```
mail hans <&0 geburtstag
```

>& fd

verwendet die Datei, welche mit dem Dateideskriptor `fd` verbunden ist, als Standardausgabe.

Benutzerfreundliche Shell-Skripts weisen den Anwender auf Fehler (wie falsche Eingaben) hin. Nun wird allerdings oft beim Aufruf von Shell-Skripts die Standardausgabe umgelenkt, und damit werden auch alle für den Benutzer gedachten Dialog-Meldungen, wie z.B.

```
echo "Bitte eine der Tasten a, b oder c druecken"
```

in diese umgelenkte Standardausgabe-Datei geschrieben.

Der Benutzer wird also diese Meldung nicht am Bildschirm sehen, da sie in die entsprechende Standardausgabe-Datei geschrieben wird. Um nun explizit zu erzwingen, daß eine solche Fehlermeldung auch bei umgelenkter Standardausgabe auf **stderr** (und damit – wenn nicht explizit umgelenkt – auf den Bildschirm) geschrieben wird, werden oft folgende **echo**-Meldungen in Shell-Skripts angegeben:

```
echo "Bitte eine der Tasten a, b oder c druecken" >&2
```

<&-

schließt die Standardeingabe (identisch zu `</dev/null`).

>&-

schließt die Standardausgabe.

Wird oft anstelle von `>/dev/null` verwendet, wenn nur die Ausführung, aber nicht die Ausgabe eines Kommandos wichtig ist.

Bei all diesen Notationen kann zusätzlich vor dem entsprechenden Umlenkungszeichen noch ein Dateideskriptor (Zahl) angegeben werden, der den umzulenkenden »Datenstrom« (wie z.B. 2 für Standardfehlerausgabe) festlegt:

Beispiel

`2>fehler`

lenkt die Standardfehlerausgabe in die Datei *fehler* um

`2>>fehl_sammel`

lenkt die Standardfehlerausgabe in die Datei *fehl_sammel* um, wobei die entsprechende Fehlermeldung allerdings an das Ende der Datei *fehl_sammel* geschrieben wird.

`2>&-`

schließt die Standardfehlerausgabe.

`2>&1`

lenkt die Standardfehlerausgabe in die Datei mit dem Dateideskriptor 1 (Standardausgabe) um.

`0<eingab_dat`

ist identisch zur Angabe `<eingab_dat`.

`auswert 1>sammel 2>&1`

verbindet zunächst Standardausgabe (Dateideskriptor 1) mit der Datei *sammel*; dann wird die Standardfehlerausgabe (Dateideskriptor 2) in die Datei, die mit Dateideskriptor 1 (*sammel*) verbunden ist, umgelenkt.

Hinweise

1. Vor und nach den Umlenkungsanweisungen können beliebig viele Leer- und Tabulatorzeichen angegeben sein, wie z.B.:
`cat >     aus <     ein`
2. Die Reihenfolge, in welcher die »Umlenkungsanweisungen« angegeben sind, ist signifikant: Die Shell wertet immer »von links nach rechts aus«.

Beispiel

Bei der Eingabe hat sich jemand vertippt und anstatt den Fehler zu korrigieren, wird einfach eine neue Umlenkung angegeben.

```
$ echo "Hallo Egon" >gruss_egon ↵
$ cat gruss_egon ↵
Hallo Egon
$ echo "Hallo Hans" >gruss_egon >gruss_hans ↵
$ cat gruss_hans ↵
Hallo Hans
$ cat gruss_egon ↵
$
```

Die Datei *gruss_egon* ist leer. Der Grund hierfür ist: Zunächst wird die Standardausgabe in die Datei *gruss_egon* umgelenkt, d.h., der Inhalt dieser Datei wird gelöscht. Das spätere Umlenken in die Datei *gruss_hans* bewirkt zwar, daß die Ausgabe in diese Datei *gruss_hans* (und nicht nach *gruss_egon*)

erfolgt. Die Löschaktion für Datei *gruss_egon* kann jedoch nicht mehr rückgängig gemacht werden.

- Die oben vorgestellten Umlenkungs-Konstruktionen werden bereits vor dem Aufruf des entsprechenden Programms von der Shell ausgewertet, so daß das aufgerufene Programm davon keinerlei Notiz nimmt. Nachdem die Shell die geforderten Umlenkungen vorgenommen hat, werden diese Umlenkungsangaben nicht mehr benötigt und deshalb von der Shell aus der Kommandozeile entfernt. Somit werden Umlenkungsanweisungen niemals einem Programm als Argumente übergeben.

Beispiel

```
$ cat dru␣  
echo Argumentzahl: $#  
echo $1 $2  
$ chmod u+x dru␣  
$ dru hans >ausg␣  
$ cat ausg␣  
Argumentzahl: 1  
hans  
$
```

- Umlenkungsanweisungen können an beliebiger Stelle in einem einfachen Kommando angegeben werden.

Beispiel

Alle folgenden Kommandozeilen sind identisch:

```
cat <ein >aus (1)  
cat >aus <ein (2)  
<ein cat >aus (3)  
<ein >aus cat (4)  
>aus cat <ein (5)  
>aus <ein cat (6)
```

Aus Gründen der Lesbarkeit sollten die Umlenkungsangaben allerdings am Ende eines Kommandos – wie bei (1) und (2) – angegeben werden.

- Parameter- und Kommandosubstitution werden immer zuerst durchgeführt, bevor die entsprechenden Umlenkungen von der Shell vorgenommen werden:

```
$ ausg_datei="/tmp/${LOGNAME}.ausg$$"␣  
$ fehl_datei="/tmp/${LOGNAME}.fehl$$"␣  
$ ls c* >$ausg_datei␣  
$ cat $ausg_datei␣  
core  
countdown2  
countdown  
cph  
$ ls b* 2>$fehl_datei␣  
$ cat $fehl_datei␣  
b*: No such file or directory
```

```
$ od -cb <`pwd`
0000000 220 \b . \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
      220 010 056 000 000 000 000 000 000 000 000 000 000 000 00
0000020 372 007 . . \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
      372 007 056 056 000 000 000 000 000 000 000 000 000 000 000
0000040 221 \b d i v _ n u l l . c \0 \0 \0 \0
      221 010 144 151 166 137 156 165 154 154 056 143 000 000 000 000
      :
      :
      :
0001600 9 \t a u s g \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
      071 011 141 165 163 147 000 000 000 000 000 000 000 000 000
0001620
$
```

6. Im Hintergrund gestartete Kommandos verwenden, wenn keine Umlenkungsanweisungen angegeben sind, die Voreinstellungen für die Standardausgabe und Standardfehlerausgabe: die Dialogstation. Dies leuchtet ein. Was passiert jedoch bei Hintergrundjobs mit der Standardeingabe? Die Eingabe über Tastatur ist nämlich nur für Vordergrundjobs vorgesehen. Für diese wurde folgende Konvention getroffen: Bei Hintergrundjobs ist die Voreinstellung für die Standardeingabe die leere Datei **/dev/null**.

Beispiel

```
$ wc div_null.c &
1139                                [PID des Hintergrundjobs]
      9      21      129 div_null.c
$ wc &
1178                                [erwartet eine Eingabe von der Tastatur]
      0      0      0                                [PID des Hintergrundjobs]
      0      0      0                                [Eingabe unmöglich, da Standardeing. in
                                                    /dev/null umgelenkt]
$
```

7. Für die bei der Ausgabeumlenkung angegebene Datei findet keine Dateinamenexpansion statt.

Beispiel

```
$ ls so*
sortiert
$ sort namdatei >so*
$ ls so*
so*
sortiert
$ cat so\*
anton
emil
fritz
hans
$
```

8. Wenn sich ein angegebenes Kommando aus mehreren einfachen Kommandos zusammensetzt, so wertet die Shell zuerst die Umlenkungsanweisungen für das gesamte Kommando aus, bevor sie die Umlenkungsangaben für die einzelnen einfachen Kommandos auswertet. Somit ergibt sich folgende Auswertungsreihenfolge:
 - a) Auswertung für die gesamte *Kommandoliste*
 - b) Auswertung für jede einzelne *Pipeline* in der *Kommandoliste*
 - c) Auswertung für jedes einzelne Kommando in den angegebenen *Pipelines*
 - d) Auswertung für jede *Kommandoliste* in jedem Kommando
9. Eine Subshell erbt die Dateideskriptoren der aufrufenden Shell.

Beispiel

```
$ exec 2>fehler >logfile
sh [Keine Ausgabe des Promptzeichens (,da stderr umgelenkt)]
echo $$
ls so*
exit
exec 2>/dev/tty >/dev/tty
$ cat logfile
1259
so*
sortiert
$ cat fehler
$ $ echo $$
1073
$
```

10. Bei der Verwendung von **>&** ist die Reihenfolge der Angabe wichtig:

```
kdo 2>&1 1>sammel (1) Standardfehlerausgabe ---> Bildschirm
Standardausgabe ---> sammel
```

Zunächst wird hierbei die Standardfehlerausgabe (Dateideskriptor 2) auf den Bildschirm (Voreinstellung für Standardausgabe) umgelenkt; danach erst wird die Standardausgabe in die Datei *sammel* umgelenkt.

```
kdo 1>sammel 2>&1 (2) Standardausgabe ---> sammel
Standardfehlerausgabe ---> sammel
```

Solche exotischen Kommandozeilen lassen sich mit der nachfolgenden Notation nachvollziehen :

```
(2) 1 --> sammel
    2 --> 1
```

Nun wird versucht, »von unten nach oben« Weiterleitungen zu erkennen. Dies ist dann der Fall, wenn ein rechts stehender Dateideskriptor eines tiefer stehenden Ausdrucks auf der linken Seite eines höher stehenden Ausdrucks vorkommt. Dies ist hier möglich:

```
2 --> 1 1 --> sammel
2 --> 1 --> sammel
```

```
(1) 2 --> 1
    1 --> sammel
```

Nun wird versucht, »von unten nach oben« Weiterleitungen zu erkennen. Dies ist hier nicht möglich:

```
1 --> sammel 2 --> 1
nicht möglich
```

Beispiel

```
$ ls b* s* 2>&1 1>sammel
b*: No such file or directory
$ cat sammel
sammel
schiebe
so*
sortiert
stdio.h
$ ls b* s* 1>sammel 2>&1
$ cat sammel
b*: No such file or directory
sammel
schiebe
so*
sortiert
stdio.h
$ who
emil      tyib  Apr 30  10:35
egon      tyic  Apr 30  11:17
moly      tyie  Apr 30  11:28
$ who 2>&1 | wc -l
3
$
```

[Ausgabe erfolgt auf die Standardfehlerausgabe]

[Standardfehlerausgabe in Pipe umlenken]

4.10 Auswertung von Ausdrücken

Bevor in den nächsten beiden Kapiteln auf die einzelnen Programmiersprach-Konstrukte der Shell eingegangen wird, sollen zunächst die beiden Kommandos **test** und **expr** vorgestellt werden.

4.10.1 Das builtin-Kommando test

Das builtin-Kommando **test** bietet über seine große Zahl von Optionen eine Vielzahl von Möglichkeiten an, um Bedingungen zu prüfen. Die vollständige Auf-
rufsyntax für **test** ist:

test *ausdr*

oder die alternative Angabe

[*ausdr*]¹

Der angegebene Ausdruck *ausdr* wird dabei ausgewertet. Ist die über *ausdr* angegebene Bedingung erfüllt, liefert **test** bzw. [...] den exit-Status 0 (wahr oder erfolgreich), ansonsten einen von 0 verschiedenen Wert (falsch oder nicht erfolgreich). Wenn keine Argumente angegeben sind, dann liefert **test** einen exit-Status verschieden von 0 (falsch).

Es können die folgenden Ausdrücke für *ausdr* angegeben werden:

Ausdruck	liefert wahr (exit-Status 0), wenn
-r <i>datei</i>	<i>datei</i> existiert und gelesen (r ead) werden darf.
-w <i>datei</i>	<i>datei</i> existiert und beschrieben (w rite) werden darf.
-x <i>datei</i>	<i>datei</i> existiert und ausführbar (e xecute) ist.
-f <i>datei</i>	<i>datei</i> existiert und eine normale Datei (f ile) ist.
-d <i>datei</i>	<i>datei</i> existiert und ein Directory (d irectory) ist.
-c <i>datei</i>	<i>datei</i> existiert und eine »zeichenspezifische Gerätedatei« (c haracter) ist.
-b <i>datei</i>	<i>datei</i> existiert und eine »blockspezif. Gerätedatei« (b lock) ist.
-p <i>datei</i>	<i>datei</i> existiert und eine named p ipe ist.
-u <i>datei</i>	<i>datei</i> existiert und set- u ser-id-Bit gesetzt ist.
-g <i>datei</i>	<i>datei</i> existiert und set- g roup-id-Bit gesetzt ist.
-k <i>datei</i>	<i>datei</i> existiert und sticky-Bit gesetzt ist.
-s <i>datei</i>	<i>datei</i> existiert und nicht leer (s pace) ist.

Tabelle 4.5: Ausdrücke für das builtin-Kommando **test**

1. Nach [und vor] muß mindestens ein Leer- oder Tabulatorzeichen angegeben sein.

Ausdruck	liefert wahr (exit-Status 0), wenn
-t [<i>fd</i>]	die geöffnete Datei, deren Filedeskriptor <i>fd</i> ist, der Dialogstation (ter -minal) zugeordnet ist. Ist <i>fd</i> nicht angegeben, so wird hierfür der Filedeskriptor 1 angenommen.
-z <i>zkt</i>	Länge der Zeichenkette <i>zkt</i> gleich 0 (zero) ist.
-n <i>zkt</i>	Länge der Zeichenkette <i>zkt</i> nicht gleich 0 (not zero) ist.
<i>zkt1</i> = <i>zkt2</i>	Zeichenketten <i>zkt1</i> und <i>zkt2</i> identisch sind.
<i>zkt1</i> != <i>zkt2</i>	Zeichenketten <i>zkt1</i> und <i>zkt2</i> verschieden sind.
<i>zkt</i>	<i>zkt</i> nicht die leere Zeichenkette ist.
<i>n1</i> -eq <i>n2</i>	die ganzen Zahlen <i>n1</i> und <i>n2</i> gleich (equal) sind.
<i>n1</i> -ne <i>n2</i>	die ganzen Zahlen <i>n1</i> und <i>n2</i> nicht gleich (not equal) sind.
<i>n1</i> -gt <i>n2</i>	ganze Zahl <i>n1</i> größer als (greater than) die ganze Zahl <i>n2</i> ist.
<i>n1</i> -ge <i>n2</i>	ganze Zahl <i>n1</i> größer oder gleich (greater equal) der ganzen Zahl <i>n2</i> ist.
<i>n1</i> -lt <i>n2</i>	ganze Zahl <i>n1</i> kleiner als (less than) die ganze Zahl <i>n2</i> ist.
<i>n1</i> -le <i>n2</i>	ganze Zahl <i>n1</i> kleiner oder gleich (less equal) der ganzen Zahl <i>n2</i> ist.

Tabelle 4.5: Ausdrücke für das builtin-Kommando test (Fortsetzung)

Alle diese Ausdrücke können nun mit den nachfolgenden Operatoren zu neuen Ausdrücken verknüpft werden:

Ausdruck	Bedeutung
! <i>ausdr</i>	Negationsoperator (monadisch ^a): Dieser Operator negiert den Wahrheitswert des nachfolgenden Ausdrucks <i>ausdr</i> .
<i>ausdr1</i> -a <i>ausdr2</i>	AND-Operator (dyadisch ^b): Der Wahrheitswert dieses mit einem AND-Operator gebildeten Gesamtausdrucks ist nur dann wahr, wenn beide Ausdrücke <i>ausdr1</i> und <i>ausdr2</i> wahr liefern.
<i>ausdr1</i> -o <i>ausdr2</i>	OR-Operator (dyadisch): Der Wahrheitswert dieses mit einem OR-Operator gebildeten Gesamtausdrucks ist wahr, wenn mindestens einer der beiden Ausdrücke <i>ausdr1</i> oder <i>ausdr2</i> wahr liefert.
(<i>ausdr</i>)	Klammerung eines Ausdrucks: Klammern werden verwendet, um andere als die voreingestellten Prioritäten (für die Operatoren) bei der Auswertung eines Ausdrucks festzulegen.

Tabelle 4.6: Operatoren zum Verknüpfen von test-Ausdrücken

- Monadisch bedeutet, daß dieser Operator nur auf einen Operanden (Ausdruck) angewandt werden kann.
- Dyadisch bedeutet, daß ein solcher Operator immer zwei Operanden benötigt.

Die voreingestellte Prioritätsreihenfolge (höchste zuerst) ist:

```
( )  
!  
-a  
-o
```

Beispiel

```
test ! -w personaldat
```

liefert wahr, wenn für die Datei *personaldat* keine Schreiberlaubnis vorliegt.

```
test -r terminkal -a -w terminkal
```

liefert wahr, wenn die Datei *terminkal* existiert und sowohl Lese- als auch Schreiberlaubnis zugesteht.

```
test -f neuheit -o -s neuheit
```

liefert wahr, wenn es sich bei Datei *neuheit* um eine normale Datei handelt, welche nicht leer ist.

```
test ! -r terminkal -a -w terminkal
```

ist identisch zu `test ( ! -r terminkal ) -a -w terminkal`

Wenn nun der ganze Ausdruck zu negieren ist, dann ist

```
test ! ( -r terminkal -a -w terminkal )
```

anzugeben. Dieser Gesamtausdruck liefert dann wahr, wenn für *terminkal* nicht gleichzeitig Schreib- und Leseerlaubnis vorhanden ist.

Da die Shell ihre eigene Interpretation für Klammern¹ besitzt, ist für diese Quoting zu verwenden:

```
test ! \ ( -r terminkal -a -w terminkal \ )
```

Der gesamte Ausdruck könnte auch in Apostrophe oder Anführungszeichen geklammert werden:

```
test ! '( -r terminkal -a -w terminkal )'
```

```
test ! "( -r terminkal -a -w terminkal )"
```

```
$ test -x div_null.c ↵
```

```
$ echo $? ↵
```

```
1
```

```
$ test -x div_null ↵
```

```
$ echo $? ↵
```

```
0
```

```
$
```

1. Siehe Kapitel 4.11.

Hinweise

1. Jeder Operator und Ausdruck ist als eigenes Argument an das Kommando **test** zu übergeben, was bedeutet, daß die Operatoren und Ausdrücke durch Leer- und / oder Tabulatorzeichen voneinander getrennt anzugeben sind.
2. Wenn eine Datei mit **-r**, **-w** oder **-x** auf Lesbarkeit, Beschreibbarkeit oder Ausführbarkeit geprüft wird und das entsprechende Zugriffsrecht-Bit nicht für den Eigentümer gesetzt ist, dann liefert **test** einen exit-Statuswert verschiedenen von 0 (falsch), selbst wenn das entsprechende Bit für *group* oder *others* gesetzt ist.

Beispiel

```
$ chmod 077 datum ↵
$ test -r datum ↵
$ echo $? ↵
1
$
```

3. Die Operatoren **=** und **!=** haben eine höhere Priorität als die Operatoren **-r** bis **-n** (in Tabelle 4.5).
4. Da die Operatoren **=** und **!=** immer Argumente benötigen, können sie nicht bei den Operatoren **-r** bis **-n** benutzt werden.
5. Wenn mehr als ein Argument nach den Operatoren **-r** bis **-n** angegeben ist, so wird nur das erste Argument ausgewertet und die restlichen werden ignoriert, außer wenn das zweite Argument einer der Operatoren **-a** oder **-o** ist.

Beispiel

```
$ test -r sortiert datum ↵
$ echo $? ↵
0
$ test -r datum sortiert ↵
$ echo $? ↵
1
$ test -x div* ↵
$ echo $? ↵
0
$ ls -F div* ↵
div_null*      [nur dieses wurde vom vorherig. test-Kommando ausgewertet]
div_null.c
div_null2*
$
```

6. Für einen *ausdr* kann auch nur ein String angegeben werden. In diesem Fall liefert **test** WAHR (exit-Status 0), wenn dieser String nicht der Nullstring ist, ansonsten FALSCH (exit-Status 1).

```
Beispiel  $ test $nam ↵
           $ echo $? ↵
           1
           $ nam="Hans" ↵
           $ test $nam ↵
           $ echo $? ↵
           0
           $
```

4.10.2 Das Kommando **expr**

Das Kommando **expr** ermöglicht es, einfache arithmetische Berechnungen durchzuführen.

Die Aufrufsyntax für **expr** ist:

expr *argument(e)*

Die *argument(e)* stellen dabei den auszuwertenden Ausdruck dar. **expr** wertet diesen Ausdruck aus und schreibt das Ergebnis auf die Standardausgabe.

Als *argument* kann angegeben werden:

- ▶ eine Zeichenkette oder
- ▶ eine ganze Zahl oder
- ▶ eine Konstruktion, die eine ganze Zahl oder Zeichenkette liefert wie z.B. ein Variablenwert (*\$variable*) oder eine Kommandosubstitution (*`...`*) oder
- ▶ ein Operator (siehe unten).

Für die einzelnen *argument(e)* ist folgendes zu beachten:

- ▶ Die einzelnen *argument(e)* müssen durch Leer- und/oder Tabulatorzeichen voneinander getrennt angegeben werden.
- ▶ Ist ein angegebenes *argument* eine Zeichenkette, die Shell-Metazeichen beinhaltet (wie z.B. Leerzeichen oder *), muß deren Sonderbedeutung für die Shell unter Verwendung von Quoting ausgeschaltet werden.
- ▶ Ganzzahligen Argumenten kann ein Minuszeichen vorangestellt werden, um dadurch den entsprechenden Negativwert auszudrücken.
- ▶ Intern wird für ganze Zahlen die 32-Bit-Darstellung (Zweier-Komplement) verwendet.
- ▶ Liefert die Berechnung des angegebenen Ausdrucks Null, so liefert **expr** den arithmetischen Wert 0 und nicht die leere Zeichenkette.
- ▶ Wird **expr** mit nur einem Argument aufgerufen, so wird dieses Argument ausgewertet und das Ergebnis ausgegeben. Das Argument sollte keiner der unten vorgestellten Operatoren sein.

Operatoren

Die bei **expr** erlaubten Operatoren sind alle dyadische Operatoren, welche die entsprechende Berechnung mit den links und rechts vom Operator angegebenen Operanden durchführen.

Es ist zu beachten, daß

- ▶ die einzelnen Operanden und Operatoren mit Leer- und / oder Tabulatorzeichen voneinander getrennt anzugeben sind und
- ▶ gewissen Operatoren, die zugleich auch Shell-Metazeichen sind, ein `\` voranzustellen ist, um deren Sonderbedeutung für die Shell auszuschalten.

Die Tabelle 4.7 zeigt alle Operatoren, die **expr** kennt:

Operator	Bedeutung
<code>op1 : op2</code>	Der Operator : überprüft, ob der für den Operanden <code>op2</code> angegebene reguläre Ausdruck den Operanden <code>op1</code> abdeckt. Als reguläre Ausdrücke sind dabei fast alle regulären Ausdrücke von ed^a zugelassen; nur <code>^</code> hat hier nicht seine ed-Sonderbedeutung für Zeilenbeginn, da der für <code>op2</code> angegebene reguläre Ausdruck immer vom <code>op1</code>-Anfang an verglichen wird^b. Wenn der für den Operanden <code>op2</code> angegebene reguläre Ausdruck den Operanden <code>op1</code> abdeckt, so liefert der Operator : die Anzahl der abgedeckten Zeichen und sonst 0. Soll nicht die Anzahl der durch <code>op2</code> abgedeckten Zeichen in <code>op1</code>, sondern der abgedeckte Teil selbst ausgegeben werden, so ist der entsprechende reguläre Ausdruck in <code>op2</code> mit <code>\(..\)</code> zu klammern. Beispiele: <pre>\$ expr nikolaus : kola↵ 0 [kola kommt nicht am Anfang vor] \$ expr nikolaus : 'n.*a'↵ 6 \$ expr nikolaus : '.*'↵ 8 \$ expr nikolaus : '.*cola'↵ 0 \$ expr nikolaus : '.*\(.*\)'↵ laus \$ expr nikolaus : '.*\(.*\)' : lau↵ 3 \$ expr nikolaus : '\([^a]*\).*'↵ nikol \$ expr nikolaus : '\(n.*l\)us'↵ \$</pre>

Tabelle 4.7: Operatoren für das Kommando `expr`

Operator	Bedeutung
$op1 \backslash * op2$	Multiplikationsoperator: Multipliziert $op1$ mit $op2$.
$op1 / op2$	ganzzahliger Divisionsoperator: Dividiert $op1$ durch $op2$ und liefert ein ganzzahliges Ergebnis (eventuelle Nachkommastellen werden entfernt).
$op1 \% op2$	Modulooperator: Liefert den Rest aus der Ganzzahldivision von $op1$ durch $op2$. Beispiele: <pre>\$ wert1=6 \$ wert2=14 \$ nam="Peter" \$ expr 14 * 6 84 \$ expr \$wert1 / \$wert2 0 \$ expr \$wert2 / \$wert1 2 \$ expr \$wert2 \% 5 4 \$ expr \$wert1 / \$nam expr: non numeric argument \$</pre>
$op1 + op2$	Additionsoperator: Addiert $op1$ und $op2$.
$op1 - op2$	Subtraktionsoperator: Subtrahiert $op2$ von $op1$. Beispiele: <pre>\$ wert1=6 \$ wert2=14 \$ expr 14 + 6 20 \$ expr \$wert1 - \$wert2 -8 \$ expr \$wert2 - \$wert1 8 \$ expr 14 - \$wert1 8 \$</pre>
$op1 = op2$	prüft, ob $op1$ gleich $op2$ ist.
$op1 \backslash > op2$	prüft, ob $op1$ größer als $op2$ ist.
$op1 \backslash \geq op2$	prüft, ob $op1$ größer oder gleich $op2$ ist.
$op1 \backslash < op2$	prüft, ob $op1$ kleiner als $op2$ ist.
$op1 \backslash \leq op2$	prüft, ob $op1$ kleiner oder gleich $op2$ ist.
$op1 != op2$	prüft, ob $op1$ ungleich $op2$ ist.

Tabelle 4.7: Operatoren für das Kommando `expr` (Fortsetzung)

Operator	Bedeutung
	Beispiele: <pre>\$ wert1=6 \$ wert2=8 \$ nam="Peter" \$ expr \$wert1 = 6 1 [wahr] \$ expr \$wert1 = \$wert2 0 [falsch] \$ expr fritz \> franz 1 [wahr] \$ expr \$nam \< "Petersilie" 1 [wahr] \$</pre> Wenn bei den obigen Operatoren die angegebene Bedingung erfüllt ist, so wird als Vergleichsergebnis 1 und ansonsten 0 geliefert
<i>op1 \& op2</i>	liefert als Ergebnis <i>op1</i>, wenn keiner der beiden Operanden <i>op1</i> oder <i>op2</i> der Wert 0 oder die leere Zeichenkette ist, ansonsten wird als Ergebnis 0 geliefert. Beispiele: <pre>\$ NAME=hans \$ expr \$NAME \& nachname hans \$ expr "\$VORNAME" \& "Nachname ?" 0 \$ expr \$VORNAME \& nachname expr: syntax error \$</pre> Die letzte Kommandozeile liefert einen Syntaxfehler, da <i>VORNAME</i> undefiniert ist, so daß expr mit <pre>expr \& "Nachname ?"</pre> aufgerufen wird, und somit der linke Operand zu \& fehlt.
<i>op1 \ op2</i>	liefert als Ergebnis <i>op1</i>, wenn der Operand <i>op1</i> nicht der Wert 0 oder die leere Zeichenkette ist, ansonsten wird als Ergebnis <i>op2</i> geliefert. Beispiele: <pre>\$ tmpdir=`expr "\$TEMP" \  "/tmp"` \$ echo \$tmpdir /tmp \$ TEMP=\$HOME/shellueb \$ tmpdir=`expr "\$TEMP" \  "/tmp"` \$ echo \$tmpdir /user1/egon/shellueb \$</pre>

Tabelle 4.7: Operatoren für das Kommando *expr* (Fortsetzung)

- Siehe erstes Buch "Linux/Unix-Grundlagen".
- op2* ist so gesehen immer die Kurzform für $\wedge op2$.
- Entsprechend C-Konvention.

Die folgende Tabelle gibt die Prioritätsreihenfolge der Operatoren in abnehmender Priorität (höchste Priorität zuerst) an. Alle in einer Zeile angegebenen Operatoren besitzen die gleiche Priorität:

```

:
\*  /      %
+   -
=   \>     \>=    \<    \<=    !=
\&
\|

```

Beispiel

```

$ expr "Hallo"
Hallo
$ expr 4+5 [kein Trennzeichen zw. 4, + und 5; als ein String
            interpretiert]
4+5 [wird als ein String ausgegeben]
$ expr 4 + 5 \* 3
19
$ expr `expr 4 + 5` \* 3
27
$

```

`zaehl=`expr $zaehl + 1``
addiert 1 auf die Shell-Variable *zaehl*.

`zeich_zahl=`expr "$NAME" : '.*'``
bestimmt die Anzahl der Zeichen, die in der Shell-Variablen *NAME* gespeichert sind, und weist diese Zahl der Variablen *zeich_zahl* zu.

`expr `pwd` = $HOME/shellueb`
gibt 1 aus, wenn das Working-Directory */user1/egon/shellueb* ist; in allen anderen Fällen wird 0 ausgegeben.

```

$ cat basisname1
expr $1 : '.*\/(.*\)' \| $1
$ chmod u+x basisname1
$ basisname1 /user1/egon/shellueb
shellueb
$ basisname1 /
expr: syntax error
$

```

Das letzte Kommando liefert einen Syntaxfehler, da dieser Aufruf des Shell-Skripts *basisname1* zu folgendem **expr**-Aufruf führt:

```
expr / : '.*\/(.*\)' \| $1
```

und **expr** für den ersten Slash den Divisionsoperator / annimmt, zu dem dann keine gültigen Operanden vorhanden sind.

Besser wäre deshalb folgendes Shell-Skript:

```
$ cat basename2
expr /$1 : '.*\/\(.*\)'
$ chmod u+x basename2
$ basename2 /user1/egon/shellueb
shellueb
$ basename2 /
$
```

Die wirklichen Kommandos **basename** bzw. **dirname** in */bin* sind noch etwas komplexer. Für Skript-Tüftler werden hier mögliche Realisierungen für diese beiden Kommandos gegeben:

```
# basename:
expr \
    "/${1:-.}" : '\(..*\/\)*$' : '.*\/\(.*\)' : "\(\..*\)$2\$" \ | \
    "/${1:-.}" : '\(..*\/\)*$' : '.*\/\(.*\)' \ | \
    "/${1:-.}" : '.*\/\(.*\)'

# dirname:
expr \
    "${1:-.}/" : '\(/\/\)*\[^\]*\[^\]*$' \ | \
    "${1:-.}/" : '\(..*\/\)*\[^\]*\[^\]*$' \ | \
    .
```

Neben der Ausgabe des Ergebnisses liefert **expr** auch einen exit-Status:

- 0 wenn der angegebene Ausdruck weder 0 noch die leere Zeichenkette ist,
- 1 wenn der angegebene Ausdruck entweder 0 oder die leere Zeichenkette ist,
- 2 wenn ein unerlaubter Ausdruck angegeben wurde.

Hinweis

Nach der Bearbeitung der angegebenen Argumente durch die Shell kann **expr** Operatoren und Operanden nur noch durch die Darstellung unterscheiden. Somit ist die Angabe von Operatoren-Zeichen, die von **expr** als Operanden zu behandeln sind, nicht so einfach möglich.

Beispiel

```
$ var+=
$ expr $var = '+'
expr: syntax error
$
```


Die letzte Kommandozeile liefert einen Syntaxfehler, da nach der Bearbeitung durch die Shell folgender Aufruf von **expr** stattfinden würde:

```
expr + = +
```

expr würden also drei Operatoren ohne Operanden übergeben.

Mit Hilfe eines kleinen Tricks kann jedoch dieses Dilemma umgangen werden: Den als Operanden zu behandelnden Operatoren-Zeichen wird ein anderes Zeichen vorangestellt, so daß **expr** hierfür keine Operator-Klassifikation treffen kann und somit die gewünschte Auswertung vornimmt:

```
$ expr a$var = a'+' [daraus wird: expr a+ = a+]
1
$
```

4.11 Kommandoklammerung

Die Shell kennt zwei Arten von Kommandoklammerung:

- ▶ Runde Klammern: (*kdoliste*) und
- ▶ Geschweifte Klammern: { *kdoliste*; }

4.11.1 Runde Klammern: (*kdoliste*)

Diese Klammerung bewirkt, daß für alle in *kdoliste* angegebenen Kommandos eine Subshell gestartet wird. Dies ist vor allen Dingen dann wichtig,

- ▶ wenn einzelne Kommandos nicht im aktuellen Working-Directory ausgeführt werden sollen oder
- ▶ wenn die Shell-Variablen der aktuellen Shell nicht wegen eines Skript-Aufrufs geändert werden sollen.

Beispiel

```
$ pwd
/user1/egon/shellueb
$ cd /bin; ls a*
acctcom
ar
as
$ pwd
/bin
$ cd /user1/egon/shellueb [Rückkehr ins Working-Directory notwendig]
$
```

Mit der Klammerung der entsprechenden Kommandos wäre es nun möglich, nur für die Dauer der Ausführung des Kommandos **ls** in das Directory */bin* zu wechseln:

```
$ pwd
/user1/egon/shellueb
$ (cd /bin; ls a*)
acctcom
ar
as
$ pwd
/user1/egon/shellueb
$
```

Alle Kommandos innerhalb des Klammersnpaares (..) werden von einer Subshell ausgeführt. Deshalb wird nur in dieser Subshell zum Directory */bin* gewechselt, während das Working-Directory der dazugehörigen Elternshell unverändert bleibt.

Das nachfolgende Shell-Skript *umrech* macht vom UNIX-Kommando **bc** Gebrauch. **bc** ist eine Art Taschenrechner mit beliebig großer Genauigkeit, dem die zu berechnenden Ausdrücke über die Standardeingabe mitgeteilt werden können. Das Ergebnis schreibt **bc** dann auf die Standardausgabe. In Kapitel 9.2 wird dieses Kommando ausführlich beschrieben:

```
$ ESKALA=cm
$ ASKALA=inch
$ FAKTOR=0.3937
$ export ESKALA ASKALA FAKTOR
$ cat umrech
echo "Gib Groesse in $ESKALA ein"
read groesse
ergeb=`bc <<xxxx
    $groesse * $FAKTOR
xxxx`
echo "$groesse $ESKALA = $ergeb $ASKALA"
$ chmod u+x umrech
$ ( ESKALA=Gallonen
> ASKALA=Liter
> FAKTOR=4.5459
> umrech)
Gib Groesse in Gallonen ein
2.5
2.5 Gallonen = 11.3647 Liter
$ umrech
Gib Groesse in cm ein      [Ursprüngl. Var.werte durch vorherig. Aufruf nicht
                             verändert]

100
100 cm = 39.3700 inch
$
```

Angaben wie **&** oder Umlenkungsanweisungen beziehen sich immer nur auf das jeweilige Teilkommando. Mit der Verwendung von Klammern können nun durch den Benutzer andere als die voreingestellten Prioritäten festgelegt werden.

Beispiel `$ echo "Dateien im Working-Directory: `pwd`"; ls -C >hier_dat`
Dateien im Working-Directory: /user1/egon/shellueb
\$

In diesem Fall würde also nur die Ausgabe von **ls** in die Datei *hier_dat* umgelenkt. Die Ausgabe des **echo**-Kommandos dagegen nicht. Mit der Verwendung von Klammern könnte nun erreicht werden, daß beide Ausgaben in die Datei *hier_dat* geschrieben würden:

```
$ ( echo "Dateien im Working-Directory: `pwd`"
> ls -C) >hier_dat
$ cat hier_dat
Dateien im Working-Directory: /user1/egon/shellueb
abc          basisname2  dru          hier_dat     sammel       wd_inhalt
.....
.....
basisname1   div_null2    hallo.c      pd           var_wert
$
```

Es sind alle Dateien in den Directories */bin* und */usr/bin* zu zählen. Der Aufruf

```
$ ls /bin; ls /usr/bin | wc -l
acctcom
ar
as
....
....
who
write
149
$
```

würde diese gestellte Aufgabe nicht lösen; in diesem Fall würden nur die Dateien der Directory */usr/bin* gezählt, während die Dateien aus */bin* lediglich am Bildschirm angezeigt würden.

Die gestellte Aufgabe könnte allerdings mit dem folgenden Aufruf gelöst werden:

```
$ ( ls /bin; ls /usr/bin ) | wc -l
249
$
```

Die Ausgabe eines Textes soll für 50 Sekunden verzögert werden. Nach Ablauf dieser Zeitspanne soll

Ich habe 50 Sekunden geschlafen

ausgegeben werden.

Der Aufruf

```
sleep 50; echo "Ich habe 50 Sekunden geschlafen" &
```

wäre in diesem Fall falsch. Richtig wäre vielmehr:

```
( sleep 50; echo "Ich habe 50 Sekunden geschlafen" ) &
```

Der exit-Status der ganzen in (..) angegebenen *kdoliste* ist der exit-Status des letzten Kommandos dieser *kdoliste*.

Beispiel

```
$ ( lss ; pwd )  
lss: not found  
/user1/egon/shellueb  
$ echo $?  
0 [erfolgreich, da letztes Kommando pwd erfolgreich war]  
$ ( pwd; lss )  
/user1/egon/shellueb  
lss: not found  
$ echo $?  
1 [nicht erfolgreich, da letztes Kommando lss nicht  
erfolgreich war]  
$
```

Hinweis

Geschachtelte Klammerungen sind erlaubt.

Beispiel

```
$ pwd  
/user1/egon/shellueb  
$ ( cd /bin  
> ( cd /usr/bin; pwd; ls a* )  
> pwd; ls a* )  
/usr/bin  
admin  
asa  
at  
awk  
/bin  
acctcom  
ar  
as  
$ pwd  
/user1/egon/shellueb  
$
```

4.11.2 Geschweifte Klammern: { *kdoliste*; }

Im Unterschied zur (..)-Klammerung wird hier die angegebene *kdoliste* nicht von einer neuen Subshell, sondern von der momentan aktiven Shell ausgeführt.

In den meisten Fällen macht es keinen Unterschied, ob eine *kdoliste* von der momentan aktiven Shell oder von einer Subshell ausgeführt wird; jedoch existieren – siehe die Klammerung mit (..) – einige wichtige Anwendungsfälle.

Die Hauptanwendung der {..}-Klammerung liegt darin, daß ein Benutzer hiermit die für gewisse Shell-Konstruktionen wie Pipe- oder Umlenkungsangaben vor-eingestellte Reihenfolge der Abarbeitung beeinflussen kann; allerdings sollte er hierzu nur dann diese Klammerungsart verwenden, wenn er die entsprechenden Kommandos von der aktuellen Shell ausführen lassen möchte.

Bei der Klammerung mit {..} sind zwei Syntaxregeln zu beachten:

1. Das letzte Zeichen in {..} muß ein ; (Semikolon) oder ein Neuezeile-Zeichen sein.

Beispiel

```
$ { ls /bin; ls /usr/bin;} | wc -l ↵
249
$
oder
$ { ls /bin ↵
> ls /usr/bin ↵
> } | wc -l ↵
249
$
```

2. Da es sich bei { um ein Schlüsselwort der Shell handelt, ist vor und nach { mindestens ein Trennzeichen (Leer-, Tabulator-, Neuezeile-Zeichen) anzugeben, damit die Shell es als solches erkennen kann.

Der exit-Status der ganzen in {..} angegebenen *kdoliste* ist der exit-Status des letzten Kommandos dieser *kdoliste*.

Beispiel

```
$ { lss ; pwd;} ↵
lss: not found
/user1/egon/shellueb
$ echo $? ↵
0 [erfolgreich, da letztes Kommando pwd erfolgreich war]
$ { pwd; lss;} ↵
/user1/egon/shellueb
lss: not found
$ echo $? ↵
1 [nicht erfolgreich, da letztes Kommando lss nicht
erfolgreich war]
$
```

Hinweis

Geschachtelte Klammerungen sind erlaubt.

Beispiel

```
$ pwd ↵
/user1/egon/shellueb
$ { cd /bin ↵
> { cd /usr/bin; pwd; ls a*; } ↵
> pwd; ls a*;} ↵
/usr/bin
```

```
admin
asa
at
awk
/usr/bin      [Warum diese im Vergleich zur ()-Klammerung veränderte Ausgabe ?]
admin
asa
at
awk
$ pwd
/usr/bin
$ cd
$ cd shellueb
$
```

Diese Art der Klammerung wird meist bei Funktionsdefinitionen (siehe Kapitel 4.12.6) verwendet und entspricht dabei in etwa einer **begin-end**-Blockbegrenzung in höheren Programmiersprachen.

4.12 Programmiersprachkonstrukte der Shell

Die Bourne-Shell bietet folgende Konstrukte zur Ablaufsteuerung an¹:

```
if if_kdoliste1
then
    then_kdoliste1
[ elif if_kdoliste2
    then
        then_kdoliste2 ]
:
:
[ else
    else_kdoliste ]
fi
```

1. [...] bedeutet, daß dieser Teil optional ist.

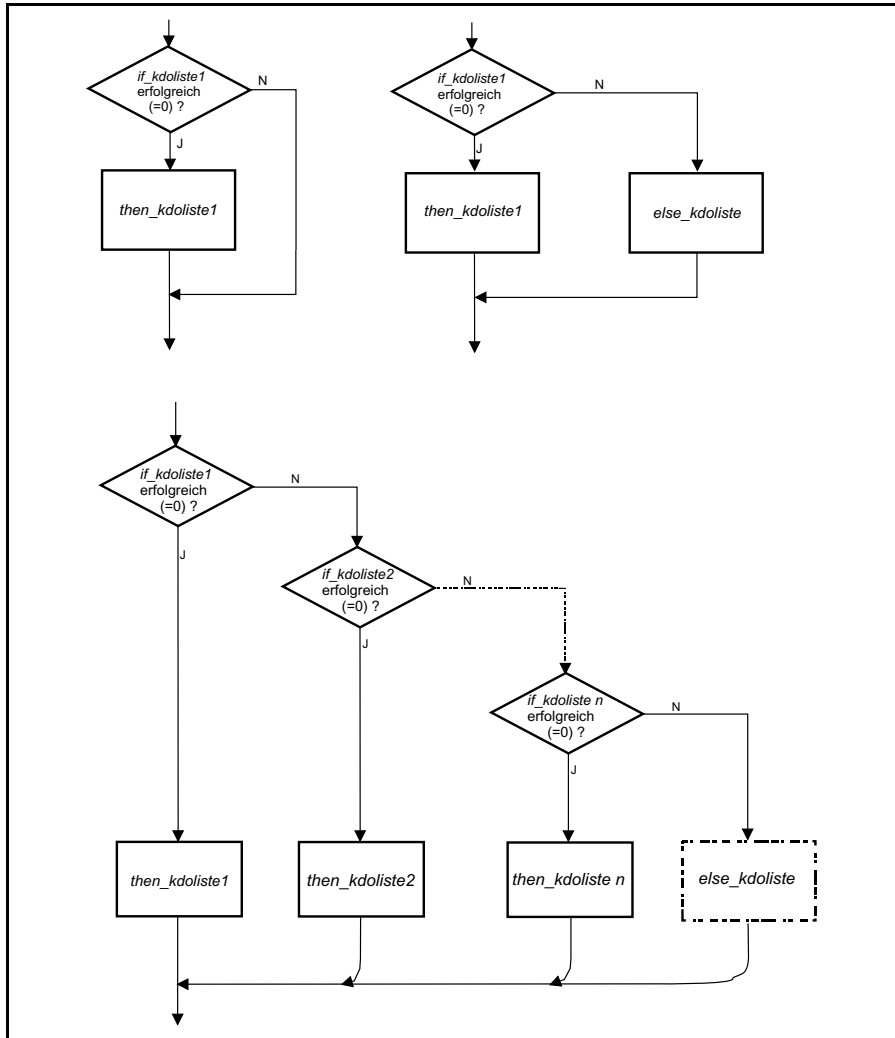


Abbildung 4.14: Ablaufpläne zum if-Kommando

```

case wort in
    pattern1) kdolist1;;
    pattern2) kdolist2;;
    :
    :
    patternn) kdolistn;;
esac

```

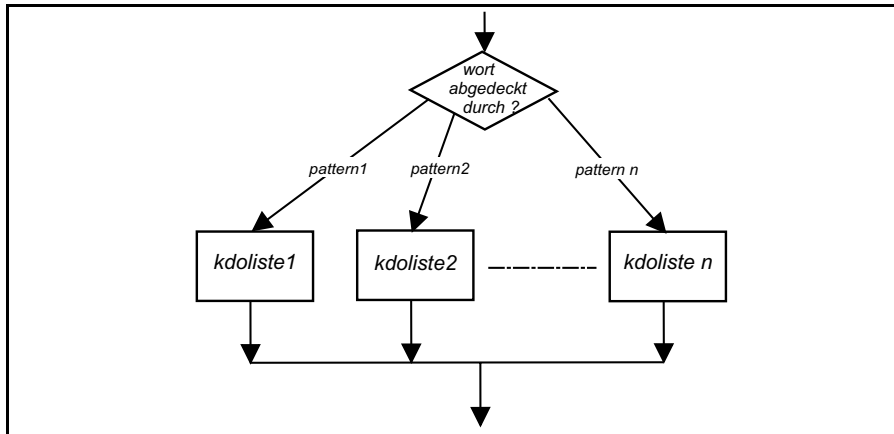


Abbildung 4.15: Ablaufplan zum case-Kommando

```

while kdoliste1
do
    kdoliste2
done
    
```

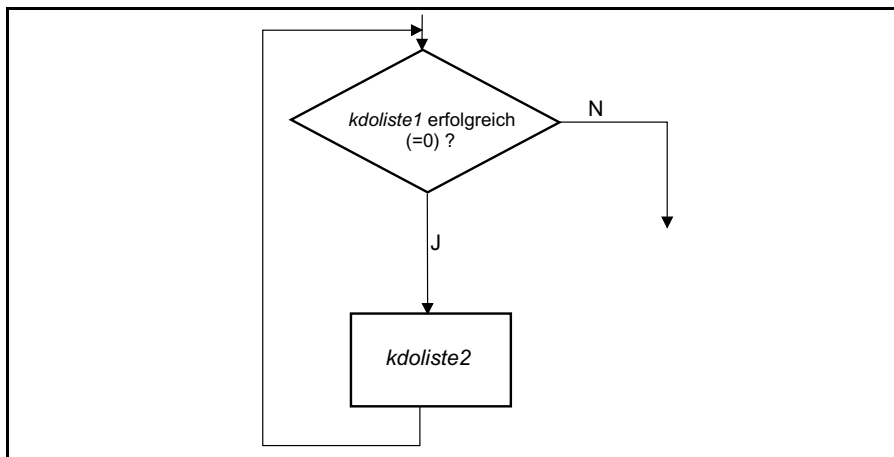


Abbildung 4.16: Ablaufplan zum while-Kommando

```

until kdoliste1
do
    kdoliste2
done
    
```

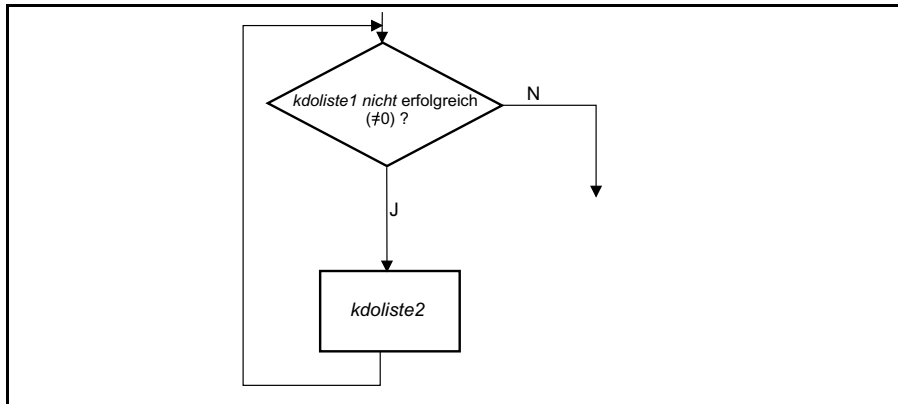



Abbildung 4.17: Ablaufplan zum until-Kommando

```

for laufvariable [ in wort1 ... wortn ]
do
    kdoliste
done
  
```

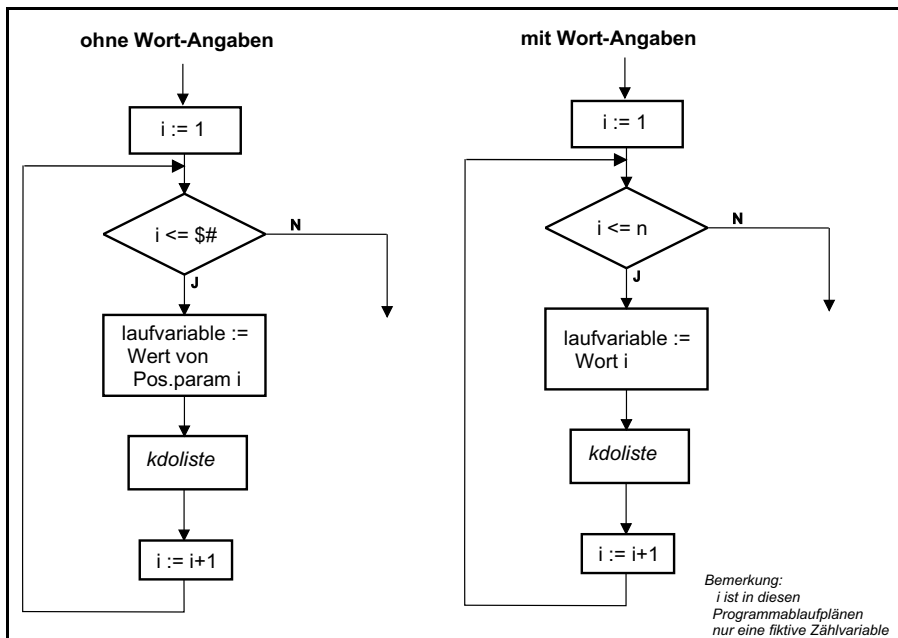


Abbildung 4.18: Ablaufplan zum for-Kommando

Funktionen

```
funktionsname() { kdoliste; }
```

Damit wird der Funktion *funktionsname* die *kdoliste* zugeordnet. Ein späterer Aufruf von *funktionsname* bewirkt dann die Ausführung dieser *kdoliste*.

Da ein Kommando entweder ein einfaches Kommando, eine Kommandoklammerung oder aber eines der oben angegebenen Konstrukte sein kann, ergibt sich folgendes Syntaxdiagramm für ein Kommando:

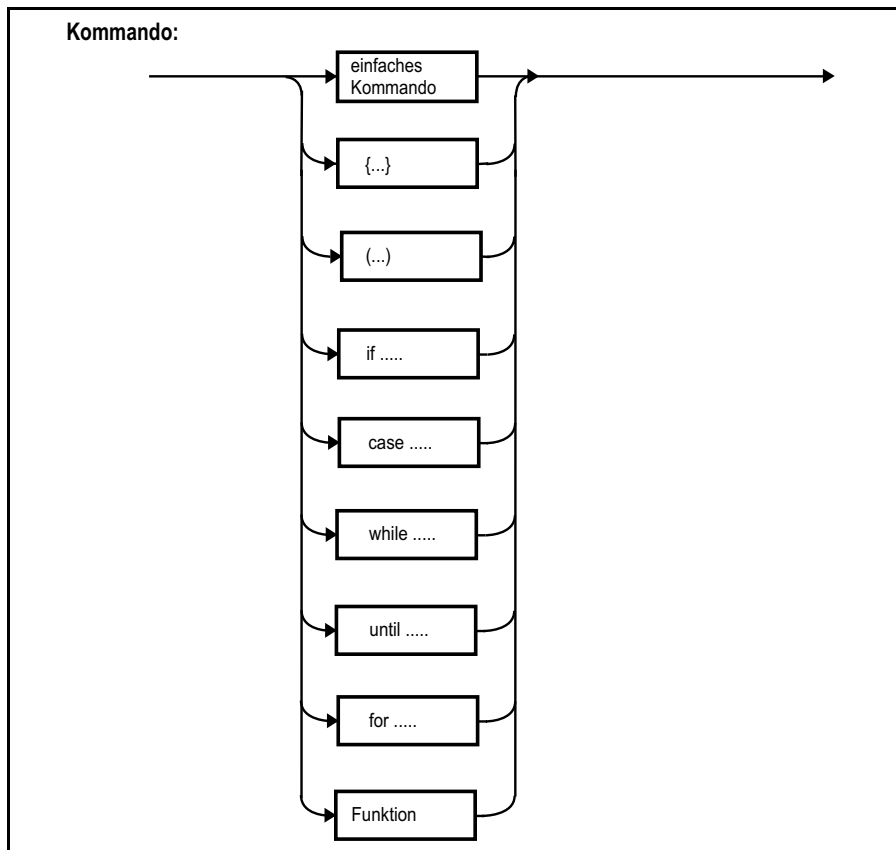


Abbildung 4.19: Syntaxdiagramm für ein Kommando

Jedes dieser Konstrukte wird als ein einziges Kommando von der Shell betrachtet; somit kann auch jedes dieser Kommandos ein Element einer Pipeline oder Liste sein.

Der exit-Status eines solchen Kommandos ist dabei, wenn nicht explizit anders beschrieben, der exit-Status des zuletzt (innerhalb des Kommandos) ausgeführten einfachen Kommandos.

Die folgenden Wörter werden von der Shell nur dann als Schlüsselwörter erkannt, wenn sie das erste Wort eines Kommandos sind und ihre Sonderbedeutung nicht durch Quoting ausgeschaltet ist:

```
if then else elif fi
case esac
for while until do done
{ }
```

Dies bedeutet, daß diese Schlüsselwörter entweder als erstes Wort einer neuen Zeile oder als erstes Wort hinter einem Semikolon anzugeben sind.

Nachfolgend werden nun die einzelnen Konstrukte zur Ablaufsteuerung genauer beschrieben.

4.12.1 Die if-Anweisung

```
if if_kdolist1
then
    then_kdolist1
[ elif if_kdolist2
    then
        then_kdolist2 ]
:
:
[ else
    else_kdolist ]
fi
```

Abhängig vom Resultat der *if_kdolist1*-Auswertung wird der Ablauf verzweigt. Dabei sind die folgenden unterschiedlichen Angaben möglich:

- ▶ **if** ohne **else**-Teil (einseitige Auswahl)
- ▶ **if** mit **else**-Teil (zweiseitige Auswahl)
- ▶ **if** mit **elif**-Teil (Mehrfach-Auswahl)

if ohne else-Teil (einseitige Auswahl)

```
if if_kdolist
then
    then_kdolist
fi
```

Zuerst werden die Kommandos aus *if_kdolist* ausgeführt: Wenn der exit-Status dieser Kommandoliste (des letzten Kommandos) gleich 0 (erfolgreich) ist, dann

werden die Kommandos aus *then_kdliste* ausgeführt; ansonsten werden diese einfach übersprungen:

Beispiel

1. Das nachfolgende Shell-Skript *vorh* meldet, ob das beim Aufruf angegebene erste Argument als Datei- oder Directoryname existiert.

```
$ cat vorh ↵
echo "$1 ist \c"          # unter Linux (bash): echo -e "... \c"
if [ ! \( -f $1 -o -d $1 \) ]
then
    echo "nicht \c"       # unter Linux (bash): echo -e "... \c"
fi
echo "vorhanden"
$ chmod u+x vorh ↵
$ vorh liste ↵
liste ist nicht vorhanden
$ vorh div_null.c ↵
div_null.c ist vorhanden
$ vorh /bin ↵
/bin ist vorhanden
$
```

2. Die nachfolgende Konstruktion überprüft, ob zumindest ein Argument beim Aufruf angegeben wurde; wenn nicht, dann wird eine Meldung ausgegeben, und das Shell-Skript mit exit-Status 1 verlassen:

```
if test $# -eq 0
then
    echo "Ein Argument muss zumindest angegeben werden"
    exit 1
fi
```

3. \$ cat compi ↵

```
if cc -o $1 $1.c
then
    echo "Datei $1.c wurde erfolgreich kompiliert" >&2
    echo "Programm $1 kann nun gestartet werden" >&2
    exit 1
fi
echo "Datei $1.c konnte nicht kompiliert werden" >&2
$ chmod u+x compi ↵
$ compi div_null ↵
Datei div_null.c wurde erfolgreich kompiliert
Programm div_null kann nun gestartet werden
$ compi dd ↵
0: No source file dd.c
Datei dd.c konnte nicht kompiliert werden
$
```

if mit else-Teil (zweiseitige Auswahl)

```
if if_kdoliste
then
    then_kdoliste
else
    else_kdoliste
fi
```

Zuerst werden die Kommandos aus *if_kdoliste* ausgeführt: Wenn der exit-Status dieser Kommandoliste (des letzten Kommandos) gleich 0 (erfolgreich) ist, dann werden die Kommandos aus *then_kdoliste*, ansonsten werden die aus *else_kdoliste* ausgeführt:

Beispiel

1. Es ist ein Shell-Skript *userda* zu erstellen, das zu einem übergebenen Login-Namen feststellt, ob dieser auf dem System existiert oder nicht:

```
$ cat userda
if cut -f1 -d":" /etc/passwd | grep "^$1$" >/dev/null 2>/dev/null
then
    echo Login-Name $1 in Passwortdatei vorhanden
    exit 0
else
    echo Login-Name $1 nicht in Passwortdatei vorhanden
    exit 1
fi
$ chmod u+x userda
$ userda egon
Login-Name egon in Passwortdatei vorhanden
$ userda eg
Login-Name eg nicht in Passwortdatei vorhanden
$
```

2. Das nachfolgende Shell-Skript *datv* vergleicht die Zeilenanzahl von zwei Dateien:

```
$ cat datv
if [ $# -ne 2 ]
then
    echo "usage: $0 datei1 datei2" >&2
    exit 1
else
    if [ `cat "$1" | wc -l` -ne `cat "$2" | wc -l` ]
    then
        echo "Zeilenzahl von $1 und $2 ist unterschiedlich" >&2
    else
        echo "Zeilenzahl von $1 und $2 ist gleich" >&2
    fi
fi
$ chmod u+x datv
$ datv div_null.c abc
```

```
Zeilenzahl von div_null.c und abc ist unterschiedlich
$ datv div_null.c hallo.arc ↵
Zeilenzahl von div_null.c und hallo.arc ist gleich
$
```

In diesem Beispiel muß der Inhalt der Dateien dem **wc**-Kommando über eine Pipe übergeben werden, da sonst **wc** bei der Ausgabe den Dateinamen mit ausgibt.

3. Das nachfolgende Skript *cpdir* kopiert einen ganzen Directorybaum. Wird die Option **-s** (*silent*) angegeben, so wird nicht – wie sonst – für jede kopierte Datei deren Name ausgegeben:

```
$ cat cpdir ↵
# cpdir   Version   v1.0   (20.11.1996)
#               kopiert den Directorybaum quelldir nach zieldir
#
# Syntax: cpdir [-s] quelldir zieldir
#           -s keine Ausgabe der Namen von kopierten Dateien
#
# Autor: Egon ...
#
if [ $# -lt 2 -o $# -gt 3 ]
then
    echo "cpdir: falsche Anzahl von Argumenten" >&2
    echo "usage: cpdir [-s] quelldir zieldir" >&2
    exit 1
fi

if [ "$1" = "-s" ]
then
    OPTION="-pd"
    shift
else
    OPTION="-pdv"
fi

QUELLE=$1
ZIEL=$2

if [ -d $ZIEL ]
then
    echo "\"$ZIEL\" existiert bereits. Loeschen ? (J/N) : \c"
    read ANTWORT
    if [ "$ANTWORT" = "j" -o "$ANTWORT" = "J" ]
    then
        rm -rf $ZIEL
        mkdir $ZIEL
    fi
else
    mkdir $ZIEL
```

```
fi

if [ "`echo $ZIEL | cut -c1`" = "/" ]
then
    cd $QUELLE
    find . -print -depth | sort | cpio $OPTION $ZIEL
else
    WORKDIR=`pwd`
    cd $QUELLE
    find . -print -depth | sort | cpio $OPTION $WORKDIR/$ZIEL
fi
$ chmod u+x cpdir
$
```

Da von nun an häufiger nützliche Skripts in den Beispielen entwickelt werden, sollten solche Shell-Skripts im Directory **\$HOME/bin** untergebracht werden, um sie dort zu sammeln:

```
$ cd [zum Home-Directory wechseln]
$ mkdir bin
$ cd shellueb
$ cp cpdir ../bin
$
```

Damit beim Aufruf eines solchen Skripts dieses immer gefunden wird, sollte dieses Directory in der Variablen **PATH** eingetragen werden. Am besten ist es, diese Variable in der Datei *.profile* (im Home-Directory) zu setzen. Deswegen ist es empfehlenswert, folgende Zeile in *.profile* einzutragen:

```
PATH=/bin:/usr/bin:$HOME/bin:.
```

Die Datei *.profile* könnte nun etwa folgendes Aussehen haben:

```
:
:
MAIL=/usr/mail/${LOGNAME:?}
MAILCHECK=0
PATH=/bin:/usr/bin:$HOME/bin:.
TERM=vt100
export TERM
tput init
```

4. Es ist ein Shell-Skript *gruesse* zu erstellen, das abhängig von der Tageszeit eine bestimmte Meldung ausgibt:

```
$ cat gruesse
stunde=`date +%H`
if [ `expr $stunde` -lt 12 ]
then
    echo "Guten Morgen"
    echo " Mit Frohmuth ans Werk"
```

```
else
  if [ `expr $stunde` -lt 18 ]
  then
    echo "Einen schoenen Nachmittag wuensche ich"
    echo " Wie waere es mit einer Kaffeepause"
  else
    echo "Guten Abend"
    echo " Es ist Zeit zu gehen"
  fi
fi
$ chmod u+x gruesse
$ cp gruesse ../bin
$
```

Mit dem folgenden Eintrag in *.profile* würde *gruesse* beim Anmelden, um 10.00 Uhr, um 14.30 Uhr und um 18.00 Uhr automatisch aufgerufen:

```
gruesse
at 1000 <<EOF 2>/dev/null
    gruesse
EOF
at 1430 <<EOF 2>/dev/null
    gruesse
EOF
at 1800 <<EOF 2>/dev/null
    gruesse
EOF
```

5. Ein **else** gehört immer zu der vorhergehenden noch nicht (mit **fi**) abgeschlossenen **if**-Anweisung. Das Shell-Skript *letztlog* schreibt die aktuelle Zeit in die Datei *\$HOME/.letztlog*. Wird beim Aufruf dieses Skripts die Option **-m** angegeben, so meldet es für den entsprechenden Benutzer die Anfangszeit der letzten UNIX-Sitzung.

```
# letztlog Version v1.0 (7.12.1996)
#          traegt letzte Login-Zeit in $HOME/.letztlog ein
#
# Syntax:  letztlog [-m]
#          -m meldet die Anfangszeit der letzten UNIX-Sitzung
#
# Autor: Egon ...
#
if [ $# -gt 1 ]
then
  echo "$0: falsche Anzahl von Argumenten" >&2
  echo "usage: letztlog [-m]" >&2
  exit 1
fi
if [ "$#" -eq "1" ]
then
  if [ "$1" = "-m" ]
```



```
then
    echo "\n\n**** Letzte Anmeldung war: `tail -1 $HOME/.letztlog`
****\n"
else
    echo "$0: unbekannte Option $1" >&2
    echo "usage: letztlog [-m]" >&2
    exit 1
fi
else
    date >>$HOME/.letztlog
fi
```

Dieses Skript *letztlog* sollte zunächst wieder ausführbar gemacht und nach *\$HOME/bin* kopiert werden. Ist dies geschehen, so empfiehlt es sich, in *.profile* folgende Aufrufe einzutragen

```
letztlog -m; letztlog
```

so daß bei jedem Anmeldevorgang zunächst der Beginn der letzten UNIX-Sitzung gemeldet wird, bevor der Beginn der jetzigen UNIX-Sitzung in die Datei *\$HOME/.letztlog* eingetragen wird.

if mit elif-Teil (Mehrfach-Auswahl)

```
if if_kdoliste1
then
    then_kdoliste1
elif if_kdoliste2
then
    then_kdoliste2
:
:
[ else
    else_kdoliste ]
fi
```

Zuerst werden die Kommandos aus der *if_kdoliste1* ausgeführt: Wenn der exit-Status dieser Kommandoliste (des letzten Kommandos) gleich 0 (erfolgreich) ist, dann werden die Kommandos aus *then_kdoliste1*, ansonsten die Kommandos aus *if_kdoliste2* ausgeführt. Wenn hierbei das letzte Kommando aus *if_kdoliste2* erfolgreich ausgeführt werden konnte, dann wird *then_kdoliste2* ausgeführt, ansonsten wird die nächste **elif**-Kommandoliste (falls vorhanden) ausgeführt, usw.; wenn hier das letzte Kommando erfolgreich war, dann wird der entsprechende **then**-Teil ausgeführt usw.

Falls die letzte angegebene **elif**-Kommandoliste nicht erfolgreich durchgeführt werden konnte, dann wird – wenn angegeben – die *else_kdoliste* ausgeführt, und sonst – beim Fehlen eines **else**-Teils – wird mit der Kommandoausführung hinter dem Schlüsselwort **fi** fortgefahren.

Beispiel

1. Es ist ein Shell-Skript *catd* zu erstellen, das Inhalte von normalen Dateien mit **cat** und Inhalte von Directories mit **od** ausgibt:

```
$ pwd
/user1/egon/shellueb
$ cat catd
if [ -d "$1" ]
then
    od -cx $1      # unter Linux od auf ein Directory nicht möglich
elif [ -f "$1" ]
then
    cat $1
else
    echo "$1 ist weder ein Directory noch eine einfache Datei" >&2
    exit 1
fi
$ chmod u+x catd
$ catd $HOME/bin
0000000  094f  002e  0000  0000  0000  0000  0000  0000
          0 \t  . \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000020  07fa  2e2e  0000  0000  0000  0000  0000  0000
          372 007  . . \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
0000040  0950  7063  6964  0072  0000  0000  0000  0000
          P \t  c  p  d  i  r \0 \0 \0 \0 \0 \0 \0 \0
0000060  0951  7267  6575  7373  0065  0000  0000  0000
          Q \t  g  r  u  e  s  s  e \0 \0 \0 \0 \0 \0
0000100  09ea  656c  7a74  6c74  676f  0000  0000  0000
          352 \t  l  e  t  z  t  l  o  g \0 \0 \0 \0 \0 \0
0000120
$ catd hallo.c
main()
{
    printf("Hallo egon\n");
}
$ catd data
data ist weder ein Directory noch eine einfache Datei
$
```

2. Es ist ein Shell-Skript *ampel1* zu erstellen, das für eine Ampelfarbe die erforderliche Reaktion eines Autofahrers ausgibt. Die Ampelfarbe wird dabei als erstes Argument übergeben:

```
$ cat ampel1
if [ "$1" = "gruen" ]
then
    echo "Fahren"
elif [ "$1" = "gelb" ]
then
    echo "Wenn rot -> gelb: Vorsichtig fahren"
    echo "Wenn gruen -> gelb: Stoppen"
elif [ "$1" = "rot" ]
```

```
then
    echo "Stoppen"
else
    echo "Ampel ausser Betrieb"
fi
$ chmod u+x ampell
$ ampell gelb
Wenn rot -> gelb: Vorsichtig fahren
Wenn gruen -> gelb: Stoppen
$ ampell gruen
Fahren
$ ampell orange
Ampel ausser Betrieb
$ ampell
Ampel ausser Betrieb
$
```

3. Es ist ein Shell-Skript *copy* zu erstellen, das ähnlich dem MSDOS-Kommando **copy** arbeitet.

```
$ cat copy
# copy Version v1.0 (7.12.1996)
#           simuliert das MS-DOS Kommando copy
#
# Syntax: copy quell-datei(en) [ziel-datei/directory]
#
# Autor: Egon ...
#

if test $# -eq 0
then
    echo "$0: falsche Anzahl von Argumenten" >&2
    echo "usage: copy quell-datei(en) [ziel]" >&2
    exit 1
elif test $# -eq 1
then
    cp $1 .
elif test $# -eq 2
then
    if test -f $2
    then
        echo "$2 wuerde durch diesen Aufruf: " >&2
        echo "    copy $1 $2 ueberschrieben" >&2
        echo "Ist dies beabsichtigt ? (j/n) : \c" >&2
        read antw
        if test "$antw" = "j" -o "$antw" = "J"
        then
            cp $1 $2
        else
            cp $* .
        fi
    fi
```

```

        else
            cp $1 $2
        fi
    else
        argumente="$*"
        shift `expr $# - 1`  # letztes Argument nach $1 schieben
        if test -f $1
        then
            cp $argumente .
        else
            cp $argumente
        fi
    fi
$ chmod u+x copy
$ cp copy ../bin
$

```

4. Es ist ein Shell-Skript *loesch* anzugeben, welches den als erstes Argument übergebenen Dateinamen löscht. Handelt es sich bei diesem Namen um eine Directory, so ist nach Rückfrage der ganze Directorybaum zu löschen. In jedem Fall soll der Benutzer vor dem Löschen immer gefragt werden, ob die entsprechende Datei wirklich zu löschen ist (Option **-i** beim Kommando **rm**).

```

$ cat loesch
if [ $# -ne 1 ]
then
    echo "$0: falsche Anzahl von Argumenten" >&2
    echo "usage: loesch dateiname|directoryname" >&2
    exit 1
elif [ -f "$1" ]
then
    rm -i $1
elif [ -d "$1" ]
then
    echo "*****"
    ganzen Directorybaum loeschen (g)
    nicht loeschen (n)
    *****

    Was ist zu tun : \c"
    read auswahl
    if [ "$auswahl" = "g" -o "$auswahl" = "G" ]
    then
        rm -ri $1
    fi
else
    echo "$1 ist weder ein Directory noch eine regulaere Datei" >&2
    exit 1
fi
$ chmod u+x loesch
$

```

Hinweise

1. Die einseitige **if**-Anweisung kann auch unter Verwendung von **&&** und **||** nachgebildet werden.

```
if_kdoliste && then_kdoliste
```

entspricht

```
if if_kdoliste
then
    then_kdoliste
fi
```

Beispiel

```
$ cat cc2
cc -o $1 $1.c && { echo "Datei $1.c erfolgreich kompiliert"; exit 0;}
echo "Datei $1.c nicht erfolgreich kompiliert"
exit 1
$ chmod u+x cc2
$ cc2 hallo
Datei hallo.c erfolgreich kompiliert
$ echo $?
0
$ cc2 hallo2
0: No source file hallo2.c
Datei hallo2.c nicht erfolgreich kompiliert
$ echo $?
1
$
```

```
if_kdoliste || then_kdoliste
```

entspricht

```
if ! if_kdoliste
then
    then_kdoliste
fi
```

Beispiel

```
[ $# -eq 2 ] || { echo "Mind. 2 Argumente sind anzugeben" >&2; exit 1;}
```

2. Der exit-Wert eines **if**-Kommandos ist der exit-Wert des letzten ausgeführten Kommandos (im **then**- bzw. **else**-Teil). Wenn keine Kommandos aus einem **then**- oder **else**-Teil ausgeführt wurden, dann ist der exit-Status 0 (entspricht erfolgreichem Ablauf des **if**-Kommandos).

Beispiel

```
$ if lss
> then
> echo "Ende"
> fi
lss: not found
$ echo $?
0
$
```

- Die Schlüsselwörter **if**, **then**, **else**, **elif** und **fi** müssen entweder als erstes Wort einer neuen Zeile oder aber als erstes Wort hinter einem Semikolon angegeben werden.

Beispiel

Das nachfolgende Shell-Skript *catd2* würde somit das gleiche leisten wie das zuvor vorgestellte Skript *catd*.

```
$ cat catd2 ↵
if [ -d "$1" ];then od -cx $1; elif [ -f "$1" ];then cat $1;else
echo "$1 ist weder ein Directory noch eine einfache Datei" >&2; exit 1;fi
$ rm catd2 ↵
$
```

Ein solcher Programmierstil ist allerdings nicht empfehlenswert, da hierdurch die Lesbarkeit von Shell-Skripts, die ohnehin schon durch die kryptische Syntax der Shell erschwert ist, nahezu vollständig verloren geht.

4.12.2 Die case-Anweisung

Mit der **case**-Anweisung bietet die Bourne-Shell die Möglichkeit der Mehrfach-Verzweigung an:

```
case word in
    pattern1) kdoliste1;;
    pattern2) kdoliste2;;
    :
    :
    patternn) kdolisten;;
esac
```

Während die vorherige **if**-Anweisung nur Ja/Nein-Verzweigungen zuläßt, ist es mit der **case**-Anweisung möglich, Mehrfach-Verzweigungen von einer einzigen Bedingung abhängig zu machen.

Die Zeichenkette *word* wird dabei in der angegebenen Reihenfolge zunächst mit *pattern1*, dann mit *pattern2* usw. verglichen, bis eine Übereinstimmung gefunden wird. Bei einer Übereinstimmung wird dann die zugehörige *kdoliste* ausgeführt und danach mit dem nächsten Kommando nach dem Schlüsselwort **esac** fortgefahren.

Das Zeichen **)** trennt die *pattern*-Angabe von der zugehörigen *kdoliste*; das Ende einer Kommandoliste muß mit **;;** angegeben werden.

Meistens wird für *word* der Wert einer Shell-Variablen angegeben. Die einzelnen *pattern* sind als Zeichenketten anzugeben, welche allerdings vom Mechanismus der Dateinamenexpansion Gebrauch machen können, so daß die Metazeichen *** ? []** die in Kapitel 4.7 besprochenen Auswirkungen haben. Einige Abweichungen von diesem Mechanismus gilt es hier allerdings zu berücksichtigen: / oder führender Punkt oder ein Punkt, welcher unmittelbar / folgt, müssen nicht explizit (wie bei der üblichen Dateinamenexpansion) in *word* angegeben sein.

Beispiel

1. Die Funktion des zuvor vorgestellten Shell-Skripts *ampel1* soll mit einer **case**-Anweisung realisiert werden:

```
$ cat ampel2
case "$1" in
  gruen) echo "Fahren";;
  gelb) echo "Wenn rot -> gelb: Vorsichtig fahren"
        echo "Wenn gruen -> gelb: Stoppen";;
  rot) echo "Stoppen";;
  *) echo "Ampel ausser Betrieb";;^
esac
$ chmod u+x ampel2
$ ampel2 gelb
Wenn rot -> gelb: Vorsichtig fahren
Wenn gruen -> gelb: Stoppen
$ ampel2 gruen
Fahren
$ ampel2 orange
Ampel ausser Betrieb
$ ampel2
Ampel ausser Betrieb
$
```

2. Das nachfolgende Shell-Skript *zeichtyp* bestimmt den Typ eines Zeichens, das ihm als Argument übergeben wird:

```
$ cat zeichtyp
if [ $# -ne 1 ]
then
  echo "$0: falsche Anzahl von Argumenten" >&2
  echo "usage: zeichtyp zeichen" >&2
  exit 1
fi

case "$1" in
  [0-9]) echo "Ziffer";;
  [a-z]) echo "Kleinbuchstabe";;
  [A-Z]) echo "Grossbuchstabe";;
  ?) echo "weder eine Ziffer noch ein Buchstabe";;
  *) echo "Fehler: Nur ein Zeichen erlaubt" >&2;;
esac
$ chmod u+x zeichtyp
$ zeichtyp x
Kleinbuchstabe
$ zeichtyp B
Grossbuchstabe
```

1. Der Wert des ersten Arguments wird durch keines der vorgegebenen Pattern (gruen, gelb, rot) abgedeckt, und es bleibt nur noch das Pattern * übrig, welches alle Zeichenketten abdeckt; das Pattern * entspricht also in etwa der *default*-Angabe in einer **switch**-Anweisung eines C-Programms.

```
$ zeichtyp 7
Ziffer
$ zeichtyp '<'
weder eine Ziffer noch ein Buchstabe
$ zeichtyp xyz
Fehler: Nur ein Zeichen erlaubt
$
```

3. Die **case**-Anweisung wird auch häufig bei der Klassifizierung von interaktiven Eingaben verwendet. Das nachfolgende Shell-Skript *del* löscht die als Argumente angegebenen Dateien. Allerdings fragt es zunächst den Benutzer, ob es alle Dateien mit oder ohne Rückfrage löschen soll. Zudem läßt es ihm auch noch die Möglichkeit, das Skript ohne jegliches Löschen von Dateien zu verlassen:

```
$ pwd
/user1/egon/shellueb
$ cat del
# del Version v1.0 (7.12.1996)
#          loescht Dateien nach einer vorgegebenen Benutzerwahl
#
# Syntax: del datei(en)
#
# Autor: Egon ...
#
if [ $# -lt 1 ]
then
    echo "$0: falsche Anzahl von Argumenten" >&2
    echo "usage: del datei(en)" >&2
    exit 1
fi

echo "Folgende Dateien sind zum Loeschen vorgesehen:
*****
$*
*****

(1) Alle diese Dateien ohne Rueckfrage loeschen
(r) Fuer jede einzelne Datei rueckfragen, ob loeschen oder nicht
(k) Keine dieser Dateien loeschen

Bitte waehlen: \c" >&2
read wahl
case $wahl in
    [1L]) /bin/rm "$@";;
    [rR]) /bin/rm -i "$@";;
    *) echo "---> Es wird keine Datei geloescht" >&2
esac
$ chmod u+x del
$ cp del ../bin
$
```


4. Das nachfolgende etwas umfangreichere Shell-Skript *gebdat* gibt zu einem Geburtsdatum, das interaktiv einzugeben ist, den zugehörigen Wochentagsnamen aus:

```
$ pwd
/user1/egon/shellueb
$ cat gebdat
# Einlesen des Geburtsdatums
echo "Bitte geben Sie Ihr Geburtsdatum (numerisch) ein" >&2
echo "      (tag monat jahr): \c" >&2
read tag monat jahr

# eventuell fuehrende Nullen entfernen
t=`expr $tag + 0`
m=`expr $monat + 0`
j=`expr $jahr + 0`

# Kalender zu entsprechenden Monat und Jahr ueber eine Pipe an erstes
# egrep (nur numerische Zeilen herausfiltern); dann ueber Pipe an
# zweites egrep (sucht Zeile, welche entspr. Tag enthaelt)
# Die einzelnen Felder der so herausgefilterten Zeile werden mit set
# den Positionsparametern zugewiesen, falls eine Zeile gefunden werden
# konnte.
zeile=`cal $m $j |
      egrep '^[ 0-9]*$' |
      egrep "[^0-9]$t$|^[^0-9]$t[^0-9]|^[^t[^0-9]|^[^t$"`
if [ "$zeile" = "" ]
then
    echo "Unerlaubtes Datum: $tag.$monat.$jahr" >&2
    exit 1
fi
set $zeile

# Letzten Positionsparameter der Variablen letzt_arg zuweisen
anzahl=$#
if [ $# -gt 1 ]
then
    shift `expr $# - 1`
fi
letzt_arg=$1

# Unterschiedliche Berechnung findet nur fuer die erste (evtl.) nicht
# siebentaegige Woche des entsprechenden Monats statt
if [ $!letzt_arg -ge 7 ]
then
    pos=`expr $!letzt_arg - $t + 7 - $anzahl`
else
    pos=`expr $!letzt_arg - $t`
fi

# Abhaengig von berechneter Tagesposition wird entspr. Wochentag
```

```
# ausgegeben
case $pos in
    0) echo "Samstag";;
    1) echo "Freitag";;
    2) echo "Donnerstag";;
    3) echo "Mittwoch";;
    4) echo "Dienstag";;
    5) echo "Montag";;
    6) echo "Sonntag";;
    *) echo "Fehler im Skript" >&2
esac
$ chmod u+x gebdat
$ gebdat
Bitte geben Sie Ihr Geburtsdatum (numerisch) ein
(tag monat jahr): 04 06 1956
Montag
$ gebdat
Bitte geben Sie Ihr Geburtsdatum (numerisch) ein
(tag monat jahr): 30 8 1990
Donnerstag
$ gebdat
Bitte geben Sie Ihr Geburtsdatum (numerisch) ein
(tag monat jahr): 1 1 1
Samstag
$ gebdat
Bitte geben Sie Ihr Geburtsdatum (numerisch) ein
(tag monat jahr): 31 2 1960
Unerlaubtes Datum: 31.2.1960
$ cp gebdat ../bin
$
```

Das Zeichen `|` kann verwendet werden, um mehrere alternative pattern für eine *kdoliste* anzugeben.

- Beispiel** 1. Das schon vorgestellte Shell-Skript *gruesse* wird hier mit einer **case**-Anweisung realisiert:

```
$ pwd
/user1/egon/shellueb
$ cat gruesse2
stunde=`date +%H`

case $stunde in
    0? | 1[01] )   echo "Guten Morgen"
                  echo " Mit Frohmut ans Werk";;
    1[2-7] )      echo "Einen schoenen Nachmittag wuensche ich"
                  echo " Wie waere es mit einer Kaffeepause";;
    * )           echo "Guten Abend"
                  echo " Es ist Zeit zu gehen";;
esac
$ chmod u+x gruesse2
$
```

2. Es ist ein Shell-Skript *hexziff* zu erstellen, welches bei Vorlage einer als Parameter übergebenen Hexaziffer die entsprechende Dezimalzahl dazu ausgibt:

```
$ cat hexziff
case $1 in
  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ) echo $1;;
  a | A ) echo 10;;
  b | B ) echo 11;;
  c | C ) echo 12;;
  d | D ) echo 13;;
  e | E ) echo 14;;
  f | F ) echo 15;;
  * ) echo "Keine erlaubte Hexadezimalziffer";;
esac
$ chmod u+x hexziff
$ hexziff 7
7
$ hexziff e
14
$ hexziff x
Keine erlaubte Hexadezimalziffer
$ hexziff abc
Keine erlaubte Hexadezimalziffer
$
```

Hinweise

1. Die Schlüsselwörter **case** und **esac** müssen entweder als erstes Wort einer neuen Zeile oder aber als erstes Wort nach einem Semikolon angegeben werden.
2. Der exit-Wert eines **case**-Kommandos ist der exit-Wert der letzten ausgeführten Anweisung oder eben 0, wenn keine der angebotenen Alternativen ausgewählt wurde.
3. Die angegebenen pattern werden in der Reihenfolge der Angabe daraufhin überprüft, ob sie das vorgegebene *wort* abdecken. Deswegen sollten z.B. niemals die »default«-Angaben als erste pattern angegeben werden:

Beispiel

```
$ cat hexziff2
case $1 in
  * ) echo "Keine erlaubte Hexadezimalziffer";;
  a | A ) echo 10;;
  b | B ) echo 11;;
  c | C ) echo 12;;
  d | D ) echo 13;;
  e | E ) echo 14;;
  f | F ) echo 15;;
  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ) echo $1;;
esac
$ chmod u+x hexziff2
$ hexziff 7
Keine erlaubte Hexadezimalziffer
```

```
$ hexziff e ↵
Keine erlaubte Hexadezimalziffer
$
```

Dieses Shell-Skript *hexziff2* gibt also immer (sogar für richtige Hexaziffern) die Meldung: »Keine erlaubte Hexadezimalziffer« aus, da das zuerst angegebene Pattern * alle möglichen Zeichen abdeckt.

4. Für *wort* wird neben Dateinamenexpansion auch Kommando- und Parametersubstitution durchgeführt.

4.12.3 Die while-Schleife

Mit dem **while**-Kommando bietet die Bourne-Shell eine Möglichkeit, Kommandolisten abhängig von einer Bedingung wiederholt auszuführen zu lassen:

```
while kdoliste1
do
    kdoliste2
done
```

Zuerst werden die Kommandos aus *kdoliste1* ausgeführt. Wenn der exit-Status dieser Kommandoliste (des letzten Kommandos) gleich 0 (erfolgreich) ist, dann werden die Kommandos aus *kdoliste2* ausgeführt. Dieser Ablauf wird wiederholt, bis die Ausführung von *kdoliste1* einen exit-Status verschieden von 0 (nicht erfolgreich) liefert. In diesem Fall wird die Abarbeitung hinter der **done**-Anweisung fortgesetzt.

Beispiel

1. Das nachfolgende Shell-Skript *emilda* meldet – solange **emil** am System arbeitet – alle 5 Minuten (300 Sekunden) »emil ist immer noch angemeldet«. Das Kommando **grep** durchsucht die Ausgabe von **who** nach dem login-Namen **emil**. Wenn dieser gefunden werden kann, dann resultiert hieraus der exit-Status 0, und die Schleife wird ausgeführt. Der Aufruf **sleep 300** bewirkt, daß an dieser Stelle 300 Sekunden (5 Minuten) gewartet wird, bevor die Kommandoausführung fortgesetzt wird:

```
$ cat emilda ↵
while who | grep emil >> /dev/null
do
    echo "emil ist immer noch angemeldet"
    sleep 300
done
$ chmod u+x emilda ↵
$
```

Da der Start dieses Skripts das Weiterarbeiten am System blockiert, ist es empfehlenswert, es im Hintergrund zu starten. Dies kann entweder beim Aufruf (mit der Angabe von **&**) festgelegt werden oder aber besser noch im Skript selbst, so daß hierfür der Aufrufer keine Sorge tragen muß:

```
$ cat emilda2[↵]
while who | grep emil >> /dev/null
do
    echo "emil ist immer noch angemeldet"
    sleep 300
done &
$ chmod u+x emilda2[↵]
$
```

Die Angabe von **&** hinter **done** bewirkt, daß die gesamte **while**-Schleife im Hintergrund ausgeführt wird, da eine **while**-Schleife (genauso wie die bereits vorgestellte **if**- und **case**-Anweisung) als ein Kommando von der Shell interpretiert wird.

2. Das nachfolgende Shell-Skript *fiba* druckt die Fibonacci-Zahlen am Bildschirm aus: Die beiden ersten Zahlen der Fibonacci-Zahlenfolge sind 1 und die weiteren Zahlen ergeben sich immer aus der Summe der beiden vorhergehenden Zahlen aus dieser Folge. Der Endwert der auszugebenden Zahlenfolge kann als erstes Argument übergeben werden. Wird *fiba* ohne ein Argument aufgerufen, so wird als Endwert 1000 angenommen:

```
$ cat fiba[↵]
ende=${1:-1000}
x=1
y=1
while [ $x -le $ende ]
do
    echo $x
    z=`expr $x + $y`
    x=$y
    y=$z
done
$ chmod u+x fiba[↵]
$ fiba 15[↵]
1
1
2
3
5
8
13
$ fiba[↵]
1
1
2
3
5
8
13
21
```

```
34
55
89
144
233
377
610
987
$
```

Hinweise

1. Die Schlüsselwörter **while**, **do** und **done** müssen entweder als erstes Wort einer neuen Zeile oder aber als erstes Wort nach einem Semikolon angegeben werden.
2. Die **while**-Schleife kann auch durch das builtin-Kommando **break** verlassen werden. Die Aufrufsyntax für dieses Kommando ist:

```
break [n]
```

Dieses Kommando **break** bewirkt das unmittelbare Verlassen der direkt umgebenden **while**-Schleife¹. Die optionale Angabe von *n* legt die Anzahl der Schachtelungstiefen fest, welche verlassen werden sollen. Wird *n* nicht angegeben, so wird nur eine Schleifenebene verlassen. Üblicherweise ist die Ausführung von **break** an eine Bedingung (in einer **if**- oder **case**-Konstruktion) geknüpft.

Beispiel

Das nachfolgende Shell-Skript *userda2* überprüft alle 5 Minuten, ob ein bestimmter Benutzer, dessen Login-Name als erstes Argument beim Aufruf anzugeben ist, noch angemeldet ist. Wird bei der Überprüfung festgestellt, daß der entsprechende Benutzer nicht mehr angemeldet ist, so wird dies gemeldet und das Shell-Skript verlassen. Bei der Realisierung dieses Skripts wird eine Endlosschleife benutzt, was gängige Praxis in der Shell-Programmierung ist². Eine Endlosschleife erreicht man unter Aufruf des Kommandos **true** (als **while**-Bedingung); dieses Kommando **true** führt keinerlei Aktionen durch, sondern liefert nur den exit-Status 0 (erfolgreich). Somit ist die **while**-Bedingung immer erfüllt. Um die **while**-Schleife zu verlassen, wird das Kommando **break** verwendet:

```
$ cat userda2 
while true
do
    if who | grep $1 >> /dev/null
    then
        echo "$1 ist immer noch angemeldet"
```

-
1. Gilt auch für die in den nächsten Kapiteln vorgestellten **until**- und **for**-Schleifen.
 2. Diese Vorgehensweise verstößt zwar gegen die Grundsätze der strukturierten Programmierung, allerdings hat sie sich so eingebürgert, daß sie zumindest jedem Shell-Programmierer bekannt sein sollte.

```
        sleep 300
    else
        echo "$1 hat sich nun abgemeldet"
        break
    fi
done &
$ chmod u+x userda
$
```

3. Da eine **while**-Schleife nur ein einziges Kommando darstellt, kann **&** – wie bereits zuvor gezeigt – nach **done** angegeben werden, um dieses »Schleifenkommando« im Hintergrund ablaufen zu lassen.

Beispiel

In diesem Beispiel hat der Benutzer **emil** dem Benutzer **egon** versprochen, ihm die Datei *addiere.c* in das Directory */tmp* zu kopieren. Das nachfolgende Shell-Skript *holedatei* überprüft nun alle 120 Sekunden, ob die Datei, deren Name als erstes Argument (z.B. */tmp/addiere.c*) zu übergeben ist, bereits existiert. Nachdem diese Datei eingetroffen ist, wird diese in das Working-Directory kopiert:

```
$ pwd
/user1/egon/shellueb
$ cat holedatei
# holedatei Version v1.0 (7.12.1996)
#      wartet auf das Eintreffen einer Datei und kopiert diese
#      Datei nach ihrem Eintreffen in das Working-Directory
#
# Syntax: holedatei dateiname
#      dateiname ist der Dateiname zu kopierenden Datei
#
# Autor: Egon ...
#
if [ $# -ne 1 ]
then
    echo "$0: falsche Anzahl von Argumenten" >&2
    echo "usage: holedatei dateiname" >&2
    exit 1
fi

while true
do
    if [ -s $1 ]
    then
        break
    else
        sleep 120
    fi
done &

wait $! # Auf das Ende der while-Schleife warten
```

```

sleep 100 # Um sicherzustellen, dass entsprechende Datei vollstaendig
          # uebertragen ist.

cp $1 .
echo "$1 befindet sich nun im Working-Directory" >&2
$ chmod u+x holedatei
$ holedatei /tmp/addiere.c &
1669                                     [PID des Hintergrund-Prozesses]
$ .....
.....                                [Weiterarbeiten im Vordergrund]
$ .....
/tmp/addiere.c befindet sich nun im Working-Directory
$ cp holedatei ../bin
$

```

In diesem Beispiel wurde das builtin-Kommando **wait** verwendet. Die Aufrufsyntax dieses Kommandos ist

```
wait [n]
```

Dieses Kommando **wait** veranlaßt die aktuelle Shell, auf die Beendigung des Kindprozesses mit der PID *n* zu warten. Würde **wait** ohne die Angabe einer PID *n* aufgerufen, so würde auf die Beendigung aller Kindprozesse gewartet; dies wäre hier auch möglich gewesen.

- Der exit-Status einer **while**-Schleife hängt davon ab, ob der Schleifenkörper ausgeführt wurde oder nicht. Wenn der Schleifenkörper ausgeführt wurde, dann ist der exit-Status des gesamten »Schleifen-Kommandos« der exit-Status des letzten ausgeführten Kommandos im Schleifenkörper (aus *kdoliste2*). Wenn der Schleifenkörper nicht ausgeführt wird, dann ist der exit-Status gleich 0: Dies trifft nur dann zu, wenn die Auswertung von *kdoliste1* bereits beim erstenmal negativ ausfällt. Es ist hier zu beachten, daß nicht der exit-Wert von *kdoliste1* als exit-Status für das ganze »Schleifen-Kommando« verwendet wird.

Beispiel

```

$ cat gruessoft
echo "Wieoft soll ich dich gruessen ? \c"
read zahl
x=1
name=`logname`
while [ $x -le $zahl ]
do
    echo "Guten Tag, $name"
    x=`expr $x + 1`
    if [ $x -gt $zahl ]
    then
        echo "Auf Wiedersehen, $name"    # Fehler: eco statt echo
    fi
done
echo "    exit-Status der while-Schleife: $?"

```



```
$ chmod u+x gruessoft ↵
$ gruessoft ↵
Wieoft soll ich dich gruessen ? 2 ↵
Guten Tag, egon
Guten Tag, egon
gruessoft: eco: not found
    exit-Status der while-Schleife: 1
$ gruessoft ↵
Wieoft soll ich dich gruessen ? 0 ↵
    exit-Status der while-Schleife: 0
$
```

5. Manchmal können Kommandozeilen zu lang für die Verarbeitung durch die Shell werden; in solchen Fällen kann die Verwendung der **while**-Schleife einen Ausweg bieten.

Beispiel

Das nachfolgende Shell-Skript *suchtext1* sucht den als erstes Argument angegebenen Text in allen Dateien des Directorybaums, der als zweites Argument übergeben wird.

```
$ cat suchtext1 ↵
grep -sn "$1" `find "$2" -print`
$ chmod u+x suchtext1 ↵
$ suchtext1 getchar /usr/include ↵
suchtext1: /bin/grep: arg list too long
$
```

Unter Verwendung der **while**-Schleife könnte dieses Dilemma umgangen werden:

```
$ pwd ↵
/user1/egon/shellueb
$ cat suchtext ↵
find "$2" -print | while read datei
do
    grep -sn "$1" $datei "" 1
done
$ chmod u+x suchtext ↵
$ suchtext getchar /usr/include ↵
/usr/include/stdio.h:77:#define getchar()  getc(stdin)
$ cp suchtext ../bin ↵
$
```

Die Ausgabe des **find**-Kommandos wird in diesem Beispiel über eine Pipe an das **while**-Kommando weitergegeben. Das als **while**-Bedingung angegebene Kommando **read datei** liest dann Zeile für Zeile aus dieser Pipe und

-
1. Da **grep** bei einem Aufruf mit nur einer Datei nicht den Dateinamen mit ausgibt, wird hier einfach ein zweites leeres Argument angegeben, um so **grep** zu zwingen, den Dateinamen bei Auffinden eines Textes mit auszugeben.

führt für jede Zeile den **while**-Schleifenkörper (das **grep**-Kommando) aus; diese Bedingung ist solange erfüllt, wie noch eine Zeile von **find** durch die Pipe geschickt wird.

Dieses Skript kann also niemals durch eine zu lange Kommandozeile in Verlegenheit gebracht werden. Allerdings wird diese Allgemeingültigkeit mit einer etwas langsameren Verarbeitung (gegenüber dem ursprünglichen Skript *suchtext*) erkauft: **grep** wird hier für jede einzelne Datei aufgerufen, während im ersten Skript **grep** nur einmal aufgerufen wird.

4.12.4 Die until-Schleife

Mit dem **until**-Kommando bietet die Bourne-Shell eine weitere Möglichkeit, Kommandolisten abhängig von einer Bedingung wiederholt ausführen zu lassen:

```
until kdoliste1
do
    kdoliste2
done
```

Die **until**-Schleife stellt die Umkehrung zur **while**-Schleife dar: Der Schleifenrumpf (*kdoliste2*) wird solange ausgeführt wie *kdoliste1* einen exit-Wert verschiedenen von 0 (Ausführung des letzten Kommandos aus *kdoliste1* war nicht erfolgreich) liefert.

Es ergibt sich hier also folgende Situation: Zuerst werden die Kommandos in *kdoliste1* ausgeführt. Wenn das letzte Kommando aus dieser Kommandoliste nicht erfolgreich ausgeführt werden konnte (exit-Status verschieden von 0), dann werden die Kommandos aus *kdoliste2* ausgeführt. Dieser Ablauf wird wiederholt, bis das letzte Kommando in *kdoliste1* erfolgreich durchgeführt werden kann (exit-Status gleich 0). In diesem Fall wird die Abarbeitung hinter der **done**-Anweisung fortgesetzt.

Beispiel

1. Das nachfolgende Shell-Skript *wartemil* meldet – solange **emil** noch nicht am System angemeldet ist – alle 5 Minuten (300 Sekunden) »emil ist noch nicht angemeldet«. Das Kommando **grep** durchsucht die Ausgabe von **who** nach dem login-Namen **emil**. Wenn dieser gefunden werden kann, dann resultiert hieraus der exit-Status 0 und die **until**-Schleife wird verlassen:

```
$ cat wartemil 
until who | grep emil > /dev/null
do
    echo "emil ist noch nicht angemeldet"
    sleep 300
done
echo "emil hat sich angemeldet"
$ chmod u+x wartemil 
$
```

Da der Start dieses Skripts das Weiterarbeiten am System blockieren würde, ist es empfehlenswert, es im Hintergrund zu starten. Dies kann entweder beim Aufruf (mit der Angabe von **&**) festgelegt werden oder aber besser noch im Skript selbst, so daß hierfür der Aufrufer keine Sorge tragen muß:

```
$ cat wartemil2
{ until who | grep emil > /dev/null
do
    echo "emil ist noch nicht angemeldet"
    sleep 300
done &
wait $!    # Auf die Beendigung der until-Schleife warten
echo "emil hat sich angemeldet"
} &
$ chmod u+x wartemil2
$
```

Die Klammerung des ganzen Skripts mit {...} und die Angabe von **&** bewirkt, daß das gesamte Skript im Hintergrund ausgeführt wird.

2. Das nachfolgende Shell-Skript *fiba2* druckt wieder die Fibonacci-Zahlen am Bildschirm aus. Diesmal wurde es allerdings unter Verwendung der **until**-Schleife realisiert:

```
$ cat fiba2
ende=${1:-1000}
x=1
y=1
until [ $x -gt $ende ]
do
    echo $x
    z=`expr $x + $y`
    x=$y
    y=$z
done
$ chmod u+x fiba2
$
```

3. Der Benutzer **egon** hat im Home-Directory ein Subdirectory *zeildruck*. Alle Dateien, die dorthin kopiert werden, sollen innerhalb von 10 Minuten am Drucker ausgegeben werden; beim Drucken sollen allerdings auf der ersten Seite der eigene Login-Name und der Dateiname in Spruchband-Form sowie die Zeilenanzahl ausgegeben werden. Bei der Ausgabe sollen die einzelnen Zeilen numeriert werden.

Nach dem Drucken soll die Datei wieder aus dem Directory *zeildruck* entfernt werden.

Es ist nun ein Shell-Skript *zdrucke1* zu erstellen, das alle 10 Minuten überprüft, ob neue Dateien in diesem Directory *zeildruck* angekommen sind.

Wenn ja, dann sollte dieses Skript hierfür den Druckvorgang anstoßen und danach die Datei aus diesem Directory wieder entfernen:

```
$ cd
$ mkdir zeildruck
$ cd shellueb
$ cat zdrucke1
cd $HOME/zeildruck
tmp_name=/tmp/zdrucke.$$

while true
do
    until set "" `ls`; [ $# -eq 1 ] # $1 immer leer "", um zu verhindern
    do                               # dass set ohne Argum. aufgerufen wird
        banner $LOGNAME >$tmp_name
        echo "\n-----\n\n" >>$tmp_name
        banner $2 >>$tmp_name
        echo "\n-----\n" >>$tmp_name
        echo "  Zeilenzahl: `cat $2 | wc -l` \f" >>$tmp_name
        nl -ba $2 >>$tmp_name
        lp -c $tmp_name
        rm $2 $tmp_name
    done
    sleep 600
done &
$ chmod u+x zdrucke1
$
```

Bei diesem Skript könnte es nun allerdings vorkommen, daß der Druckvorgang für eine Datei bereits angestoßen wird, obwohl diese Datei noch nicht vollständig kopiert ist. Diese Gefahr besteht besonders bei umfangreichen Dateien, wo der entsprechende Dateiname zwar schon in die Directory-Datei eingetragen wurde, aber deren Inhalt noch nicht vollständig kopiert ist.

Diese Fehlfunktion kann beseitigt werden, wenn hier der Kopier- und Druckvorgang synchronisiert werden. Die einfachste Möglichkeit, dies zu erreichen, ist die Verwendung einer »Synchronisationsdatei«. Ist eine solche Datei vorhanden, so muß das Drucken der entsprechenden Datei verzögert werden, bis diese »Synchronisationsdatei« wieder entfernt ist. Somit ergäbe sich folgende verbesserte Version des obigen Shell-Skripts:

```
$ cat zdrucke1
cd $HOME/zeildruck
tmp_name=/tmp/zdrucke.$$

while true
do
    until set "" `ls`; [ $# -eq 1 ] # $1 immer leer "", um zu verhindern
    do                               # dass set ohne Argum. aufgerufen wird
        while [ -f $2.syn ] #
```

```

do                                # neu: Synchronisations-Teil
    sleep 2                       #
done                              #
banner $LOGNAME >$tmp_name
echo "\n-----\n\n" >>$tmp_name
banner $2 >>$tmp_name
echo "\n-----\n" >>$tmp_name
echo "  Zeilenzahl: `cat $2 | wc -l` \f" >>$tmp_name
nl -ba $2 >>$tmp_name
lp -c $tmp_name
rm $2 $tmp_name
done
sleep 600
done &
$ chmod u+x zdrucke
$ cp zdrucke ../bin
$ zdrucke
$

```

Dieses Skript funktioniert allerdings nur, wenn der Benutzer vor dem Kopieren der zu druckenden Datei eine solche »Synchronisations-Datei« im Directory *zeildruck* anlegen würde. Dies könnte mit den folgenden Kommandos erreicht werden:

```

$ >$HOME/zeildruck/dateiname.syn
$ cp dateiname $HOME/zeildruck
$ rm $HOME/zeildruck/dateiname.syn

```

Allerdings ist es besser, dies durch ein eigenes Shell-Skript *cpzdr* zu automatisieren, um so dem Benutzer diese Verantwortung abzunehmen:

```

$ cat cpzdr
until [ -z "$1" ]
do
    > $HOME/zeildruck/$1.syn      # Anlegen der Synchronisationsdatei
    cp $1 $HOME/zeildruck
    rm $HOME/zeildruck/$1.syn     # Entfernen der Synchronisationsdatei
    shift
done
$ chmod u+x cpzdr
$ cp cpzdr ../bin
$

```

Sollen nun eine oder mehrere Dateien entsprechend aufbereitet am Drucker ausgegeben werden, so wäre nur

```
cpzdr dateiname1 dateiname2 ...
```

aufzurufen.

Hinweise

1. Die Schlüsselwörter **until**, **do** und **done** müssen entweder als erstes Wort einer neuen Zeile oder als erstes Wort nach einem Semikolon angegeben werden.
2. Da eine **until**-Schleife (wie eine **while**-Schleife) nur ein einziges Kommando darstellt, kann **&** – wie bereits zuvor gezeigt – nach **done** angegeben werden, um dieses »Schleifenkommando« im Hintergrund ablaufen zu lassen.
3. Die **until**-Schleife kann auch wieder durch das builtin-Kommando **break** verlassen werden.

Beispiel

Das nachfolgende Shell-Skript *wartuser* überprüft alle 5 Minuten, ob ein bestimmter Benutzer, dessen Login-Name als erstes Argument beim Aufruf anzugeben ist, bereits angemeldet ist. Wird bei der Überprüfung festgestellt, daß der entsprechende Benutzer angemeldet ist, so wird dies mit dem Klingeln der Terminalglocke gemeldet und das Shell-Skript verlassen. Bei diesem Skript wird wieder eine Endlosschleife benutzt. Eine **until**-Endlosschleife erreicht man durch den Aufruf des Kommandos **false** (als **until**-Bedingung); dieses Kommando **false** führt keinerlei Aktionen durch, sondern liefert nur einen von 0 verschiedenen exit-Status (nicht erfolgreich). Somit ist die **until**-Bedingung niemals erfüllt und der **until**-Schleifenkörper wird immer ausgeführt. Um die **until**-Schleife zu verlassen, wird hier wieder das Kommando **break** verwendet:

```
$ cat wartuser ↵
if [ $# -ne 1 ]
then
    echo "usage: $0 login-name"
    exit 1
fi

cut -f1 -d: /etc/passwd | egrep "^$1$" >/dev/null
if [ $? -ne 0 ]
then
    echo "$1 ist kein Benutzer dieses Systems"
    exit 1
fi

until false
do
    if who | grep $1 >/dev/null
    then
        echo "\007$1 ist angemeldet"
        break
    else
        echo "$1 ist noch nicht angemeldet"
        sleep 300
    fi
done &
```

```
$ chmod u+x wartuser ↵  
$ cp wartuser ../bin ↵  
$
```

4. Der exit-Status einer **until**-Schleife hängt (wie bei der **while**-Schleife) davon ab, ob der Schleifenkörper ausgeführt wurde oder nicht. Wenn der Schleifenkörper ausgeführt wurde, dann ist der exit-Status des gesamten »Schleifen-Kommandos« der exit-Status des letzten ausgeführten Kommandos im Schleifenkörper (aus *kdoliste2*).

Wenn der Schleifenkörper nicht ausgeführt wird, dann ist der exit-Status gleich 0: Dies trifft nur dann zu, wenn die Auswertung von *kdoliste1* bereits beim erstenmal positiv (exit-Wert 0) ausfällt.

4.12.5 Die for-Schleife

```
for laufvariable [ in wort1 ... wortn ]  
do  
    kdoliste  
done
```

Während die Ausführung von **while**- und **until**-Schleifen an die Auswertung von Bedingungen geknüpft ist, wird bei der **for**-Schleife die Anzahl der Schleifendurchläufe durch eine bestimmte *wort*-Liste festgelegt. Als *wort*-Liste werden dabei

- ▶ entweder implizit alle Positionsparameter (keine **in**-Angabe)
- ▶ oder die danach angegebenen *worte* (bei einer **in**-Angabe)

genommen.

for ohne in-Angabe

```
for laufvariable  
do  
    kdoliste  
done
```

Diese Konstruktion erlaubt eine wiederholte Ausführung von *kdoliste*, wobei der *laufvariable* nacheinander die Werte der Positionsparameter **\$1**, ..., **\$n** zugewiesen werden (siehe Abbildung 4.20):

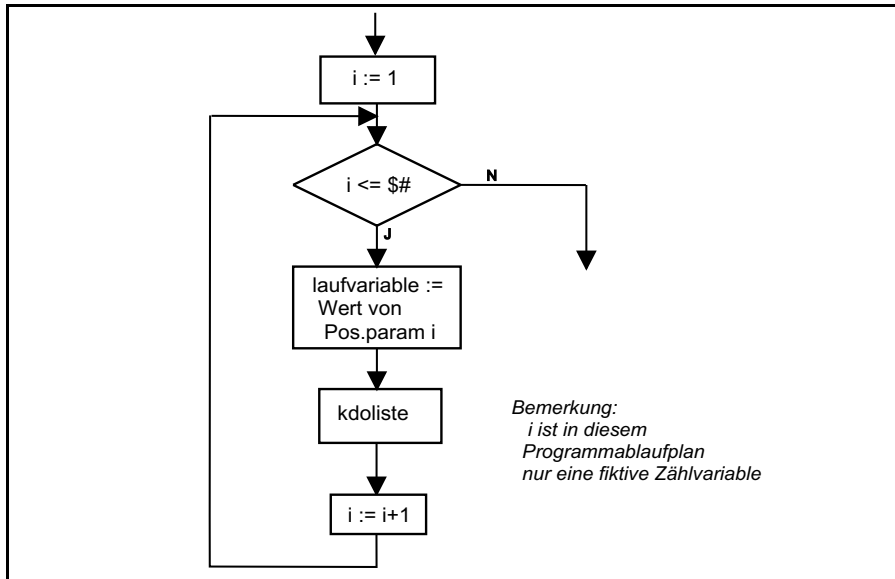


Abbildung 4.20: Ablaufplan zu for ohne in-Angabe

Beispiel

1. Das nachfolgende Shell-Skript *for1* gibt alle übergebenen Argumente mit Argument-Nummer aus:

```

$ cat for1
z=1
for j
do
    echo "$z.Argument: $j"
    z=`expr $z + 1`
done
$ chmod u+x for1
$ for1 eins zwei drei vier fuenf sechs
1.Argument: eins
2.Argument: zwei
3.Argument: drei
4.Argument: vier
5.Argument: fuenf
6.Argument: sechs
$ for1
[keine Argumente --> keine Ausgabe]
$
  
```

2. Es ist ein Shell-Skript *dateityp* zu erstellen, das für alle als Argumente übergebenen Dateinamen den Dateityp (Reguläre Datei, Directory, Block- oder Zeichenspezifische Datei) ausgibt.


```
$ cat dateityp ↵
for i
do
    echo "$i : \c"
    if [ -f $i ]
    then
        echo "regulaere Datei"
    elif [ -d $i ]
    then
        echo "directory"
    elif [ -b $i ]
    then
        echo "blockorient. Geraetedatei"
    elif [ -c $i ]
    then
        echo "zeichenorient. Geraetedatei"
    else
        echo "unbekannter Dateityp"
    fi
done
$ chmod u+x dateityp ↵
$ dateityp /dev/l* ↵
/dev/log : zeichenorient. Geraetedatei
/dev/lp : zeichenorient. Geraetedatei
/dev/lp0 : zeichenorient. Geraetedatei
/dev/lp1 : zeichenorient. Geraetedatei
/dev/lp2 : zeichenorient. Geraetedatei
$ dateityp /usr ↵
/usr : directory
$ dateityp /usr/* ↵1
/usr/adm : directory
/usr/admin : directory
/usr/bin : directory
/usr/include : directory
/usr/lib : directory
/usr/lib : directory
/usr/lost+found : directory
/usr/mail : directory
/usr/news : directory
/usr/options : directory
/usr/preserve : directory
/usr/rw.c : regulaere Datei
/usr/rw10 : regulaere Datei
/usr/rw256 : regulaere Datei
/usr/rw4096 : regulaere Datei
/usr/spool : directory
/usr/tmp : directory
$
```

1. Die Ausgabe für diesen Aufruf kann sich von System zu System unterscheiden, abhängig von den in **/usr** vorhandenen Dateien.

3. Für alle Dateinamen, die an das nachfolgende Shell-Skript *groesse* als Argumente übergeben werden, wird deren einzelner und gesamter Speicherbedarf ermittelt:

```
$ cat groesse
sum=0
for datei
do
    echo "$datei:\t\c"                # Ausgabe des Dateinamens
    if [ `expr "$datei" : '.*' -lt 7` ] # Fuer Ausgabe: Bei Dateinamen, die
    then                                # kuerzer als 7 Zeichen, zusaetzl.
        echo "\t\c"                    # Tabulator-Zeichen ausgeben:
    fi
    g=`cat "$datei" | wc -c`            # Dateigroesse bestimmen
    echo $g
    sum=`expr $sum + $g`
done

echo "-----"
echo "Gesamt:\t\t$sum"
$ chmod u+x groesse
$ groesse a*
abc: 86
abc*: 0
abc2: 88
addiere.c: 34
adress: 272
adresse.txt: 82
ampell: 242
ampel2: 208
anfang: 51
argaus: 72
argu_pid: 68
ausg: 21
ausgab: 90
-----
Gesamt: 1314
$
```

for mit in-Angabe

```
for laufvariable in wort1 ... wortn
do
    kdoliste
done
```

Diese Konstruktion erlaubt eine wiederholte Ausführung von *kdoliste*, wobei *laufvariable* nacheinander die Werte *wort1*, *wort2* ... bis *worn* annimmt (siehe Abbildung 4.21):

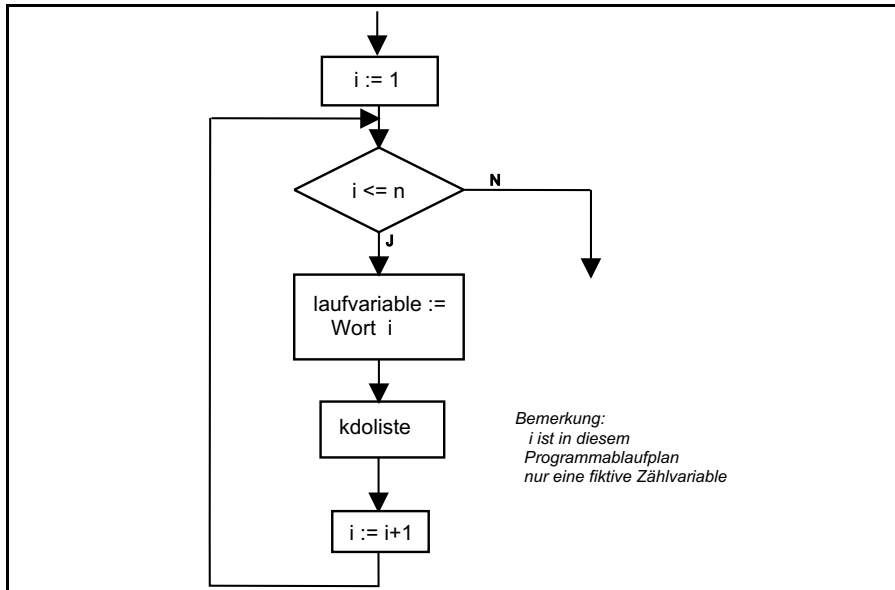


Abbildung 4.21: Ablaufplan zu for mit in-Angabe

Die vorherige Form einer **for**-Schleife (ohne **in**-Angabe) könnte also z.B. auch mit

```
for laufvariable in $@
do
    kdoliste
done
```

angegeben werden.

Beispiel

1. Das nachfolgende Shell-Skript *fahrsohn* würde die Erfolgserlebnisse eines Sohnes beim Erlernen des Fahrradfahrens ausgeben:

```
$ cat fahrsohn ↵
for i in "linken Arm" "rechten Arm" "beide Arme" "Zaehne"
do
    echo "Papa, schau ich kann schon ohne $i Fahrrad fahren"
done
$ chmod u+x fahrsohn ↵
$ fahrsohn ↵
Papa, schau ich kann schon ohne linken Arm Fahrrad fahren
Papa, schau ich kann schon ohne rechten Arm Fahrrad fahren
Papa, schau ich kann schon ohne beide Arme Fahrrad fahren
Papa, schau ich kann schon ohne Zaehne Fahrrad fahren
$
```

2. Die Shell-Skripts *cmod1* und *cmod2* setzen beide für alle Dateien im Working-Directory (außer für die, deren Name mit . beginnt) die Zugriffsmaske auf "rwxr-xr-x":

```
$ cat cmod1
for i in *
do
    chmod 755 $i
done
$ chmod u+x cmod1
$ cat cmod2
for i in `ls`
do
    chmod 755 $i
done
$ chmod u+x cmod2
$
```

3. Das nachfolgende Shell-Skript *findch* würde den als erstes Argument angegebenen Begriff in allen C-Dateien (Header- und Programmdateien) der Directorybäume suchen, die als restliche Argumente angegeben sind:

```
$ cat findch
if [ $# -lt 2 ]
then
    echo "usage: $0 begriff directory1 [directory2 ...]" >&2
    exit 1
fi

begriff=$1    # $1 in begriff sichern, und dann
shift        # aus Liste der Positionsparameter entfernen

for i in `find $* -name ".*[ch]" -print`
do
    grep -sn "$begriff" "" $i    # leerer 2.Dateiname, um grep zur Ausgabe
done                            # des Dateinamens zu zwingen
$ chmod u+x findch
$ findch TRUE /usr/include
/usr/include/sys/comm.h:30:#ifndef          TRUE
/usr/include/sys/comm.h:31:#define         TRUE 1
/usr/include/sys/fs/nfs/types.h:31:#define TRUE (1)
/usr/include/sys/slnutil.h:15:#define TRUE 1
/usr/include/sys/xtproto.h:30: *          cntl - TRUE if control packet.
/usr/include/dial.h:22:#define            TRUE 1
/usr/include/ttysrv.h:19:#define          TRUE 1
/usr/include/curses.h:82:/* TRUE and FALSE get defined so many times, */
/usr/include/curses.h:84:#if             !defined(TRUE) || ((TRUE) != 1)
/usr/include/curses.h:85:# define        TRUE (1)
$
```

4. Die nachfolgenden beiden Shell-Skripts *for2* und *for3* machen nochmals den Unterschied zwischen den beiden automatischen Variablen

\$* (alle Positionsparameter als **ein** String: "**\$1 \$2 \$3 ...**") und

\$@ (alle Positionsparameter als **einzelne** Strings: "**\$1**" "**\$2**" "**\$3**" ...) deutlich:

```
$ cat for2
echo "Argument-Anzahl: $#"  
z=1  
for arg in $*  
do  
    echo "$z.Argument: $arg"  
    z=`expr $z + 1`  
done  
$ chmod u+x for2  
$ for2 eins zwei drei  
Argument-Anzahl: 3  
1.Argument: eins  
2.Argument: zwei  
3.Argument: drei  
$ for2 'eins zwei' drei [1]  
Argument-Anzahl: 2  
1.Argument: eins  
2.Argument: zwei  
3.Argument: drei  
$
```

Bei [1] wird

for arg in \$*

durch

for arg in eins zwei drei

ersetzt (drei Worte), obwohl 'eins zwei' als ein Argument übergeben wurde. Deshalb wird die **for**-Schleife auch dreimal ausgeführt. Um zu erzwingen, daß die **for**-Schleife nur für die wirklich angegebenen Argumente durchlaufen wird, ist **\$*** durch **"\$@"** zu ersetzen:

```
$ cat for3  
echo "Argument-Anzahl: $#"  
z=1  
for arg in "$@"  
do  
    echo "$z.Argument: $arg"  
    z=`expr $z + 1`  
done  
$ chmod u+x for3  
$ for3 eins zwei drei  
Argument-Anzahl: 3
```

```
1.Argument: eins
2.Argument: zwei
3.Argument: drei
$ for3 'eins zwei' drei
Argument-Anzahl: 2
1.Argument: eins zwei
2.Argument: drei
$ for3
Argument-Anzahl: 0
$
```

5. Die Teilnehmer an einer Lottotipgemeinschaft werden aus der Datei *.lottoteil* (im Home-Directory) gelesen, und für jeden einzelnen Teilnehmer wird eine Datei *teilnehmername.lotto* angelegt, in welche zunächst der Teilnehmernamen in Spruchband-Form, dann eine Trennzeile »-----« und schließlich der Gewinn (als erstes Argument übergeben) geschrieben wird. Diese Datei wird dann am Drucker ausgegeben und danach gelöscht.

```
$ cat lottodruck
if [ $# -ne 1 ]
then
    echo "usage: $0 gewinn"
    exit 1
fi

for teilnehm in `cat $HOME/.lottoteil`
do
    banner $teilnehm >$teilnehm.lotto
    echo "-----" >>$teilnehm.lotto
    echo "\n\nGewinn in letzter Woche: $1 DM" >>$teilnehm.lotto
    lp -c $teilnehm.lotto
    rm $teilnehm.lotto
done
$ chmod u+x lottodruck
$ cat $HOME/.lottoteil
emil
egon
$
```

Würde in diesem Fall das Skript *lottodruck* mit dem Aufruf

lottodruck 12.56

gestartet, würden die folgenden beiden Seiten am Drucker ausgegeben:

```
##### # # # #
# ## ## # #
##### # ## # # #
# # # # #
# # # # #
##### # # #####
```

Gewinn in letzter Woche: 12.56 DM

```
##### #### # #
# # # # # ## #
##### # # # # #
# # ## # # # #
# # # # # # ##
##### #### # #
```

Gewinn in letzter Woche: 12.56 DM

Hinweise

1. Die Schlüsselwörter **for**, **do** und **done** müssen entweder als erstes Wort einer neuen Zeile oder als erstes Wort nach einem Semikolon angegeben werden. Einzige Ausnahme ist das Schlüsselwort **do** bei einer **for**-Schleife ohne **in**-Angabe; in diesem Falle kann nämlich **do** unmittelbar nach *laufvariable* angegeben werden.

Beispiel

Das nachfolgende Shell-Skript *kreiere* legt leere Dateien an; als Namen verwendet es dabei die übergebenen Argumente:

```
$ cat kreiere 
for i do >$i; done # Angabe von ; vor do hier nicht noetig
$ chmod u+x kreiere 
$
```

2. Da eine **for**-Schleife (wie eine **while**- oder **until**-Schleife) nur ein einziges Kommando darstellt, kann **&** – wie bei den vorherigen Schleifen gezeigt – nach **done** angegeben werden, um dieses »Schleifenkommando« im Hintergrund ablaufen zu lassen.
3. Die **for**-Schleife kann auch wieder durch das builtin-Kommando **break** verlassen werden.
4. Der exit-Status einer **for**-Schleife hängt (wie bei der **while**- oder **until**-Schleife) davon ab, ob der Schleifenkörper ausgeführt wurde oder nicht. Wenn der Schleifenkörper ausgeführt wurde, dann ist der exit-Status des gesamten »Schleifen-Kommandos« der exit-Status des letzten ausgeführten

Kommandos im Schleifenkörper. Wenn der Schleifenkörper nicht ausgeführt wird, dann ist der exit-Status gleich 0.

4.12.6 Allgemeine Regeln für die Kommandos **if**, **case**, **while**, **until** und **for**

1. Der Aufruf des builtin-Kommandos **continue** in einem Schleifen-Kommando (**while**, **until**, **for**) bewirkt die unmittelbare Ausführung des nächsten Schleifendurchlaufs. Im Gegensatz zu **break** wird also bei **continue** nicht die gesamte Schleife verlassen, sondern lediglich der Rest des Schleifenkörpers übersprungen und sofort mit dem nächsten Schleifendurchlauf fortgefahren. Die Aufrufsyntax für dieses Kommando ist:

```
continue [n]
```

Die optionale Angabe von *n* legt die Anzahl der Schachtelungstiefen fest, welche übersprungen werden sollen. Ist *n* nicht angegeben, so wird nur für die direkt umschließende Schleife der nächste Schleifendurchlauf gestartet. Üblicherweise ist die Ausführung von **continue** an eine Bedingung (in einer **if**- oder **case**-Konstruktion) geknüpft.

Beispiel

Das nachfolgende Shell-Skript *klebe* überprüft alle als Argumente übergebenen Dateinamen darauf hin, ob es sich bei der jeweiligen Datei um eine Textdatei handelt. Wenn ja, so wird deren Inhalt an die Datei *zusamm* angehängt, ansonsten nicht. Um die einzelnen Dateien voneinander abzutrennen, wird vor jeder Datei die Trennzeile "@@---Dateiname---@@" nach *zusamm* geschrieben:

```
$ cat klebe ↵
for datei
do
    file $datei | grep "text" >/dev/null 2>/dev/null
    if [ $? -ne 0 ]
    then
        echo "$datei keine Textdatei" >&2
        continue
    fi
    echo "@@---$datei---@@" >>zusamm
    cat $datei >>zusamm
done
$ chmod u+x klebe ↵
$ file di* ↵
diskuse:                                commands text
div_null:                               iAPX 386 executable not stripped
div_null.c:                             c program text
div_null2:                              iAPX 386 executable not stripped
$ klebe di* ↵
div_null keine Textdatei
div_null2 keine Textdatei
```



```
$ cat zusamm ↵
@@---diskuse---@@
du / >$HOME/.diskuse
mail $LOGNAME <<ENDE_NACHRICHT
    Die Belegungsstatistik fuer das Dateisystem /
    befindet sich in der Datei $HOME/.diskuse
ENDE_NACHRICHT
@@---div_null.c---@@
#include <stdio.h>

main()
{
    int divi, a=5, b=0;

    divi=a/b;
    printf("Die Division von %d / %d = %d\n", a, b, divi);
}
$
```

Die Funktion des Skripts *klebe* hätte natürlich auch ohne die Verwendung von **continue** realisiert werden können; das nachfolgende Skript *klebe2* ist vollständig äquivalent zu *klebe*:

```
$ cat klebe2 ↵
for datei
do
    file $datei | grep "text" >/dev/null 2>/dev/null
    if [ $? -ne 0 ]
    then
        echo "$datei keine Textdatei" >&2
    else
        echo "@@---$datei---@" >>zusamm
        cat $datei >>zusamm
    fi
done
$
```

2. Es ist möglich, Ein/Ausgabeumlenkungen für ein ganzes »Programmiersprach-Kommando« (**if**, **case**, **while**, **until** oder **for**) anzugeben, indem nach dem abschließenden Schlüsselwort (**fi**, **esac** oder **done**) die entsprechenden Umlenkungsanweisungen angegeben werden:

Die Umlenkung der Standardeingabe für ein ganzes solches Kommando bewirkt, daß alle in dieses Kommando eingebetteten Kommandos, die ihre Daten von der Standardeingabe lesen, diese nun aus der entsprechenden Datei (neue Standardeingabe) lesen.

Die Umlenkung der Standardausgabe oder Standardfehlerausgabe für ein gesamtes solches Kommando bewirkt, daß alle in dieses Kommando eingebetteten Kommandos, die Daten ausgeben, diese nun auf die entsprechende Datei (neue Standardausgabe bzw. Standardfehlerausgabe) schreiben.

Soll ein bestimmtes eingebettetes Kommando nicht mit dieser »Gesamt-Ein / Ausgabeumlenkung« gekoppelt werden, so kann dort explizit eine andere Umlenkungsvorgabe angegeben werden. So kann z.B. die Ausgabe eines bestimmten **echo**-Kommandos immer auf das Terminal erfolgen, indem

```
echo "... " >/dev/tty
```

angegeben wird.

Beispiel

Das nachfolgende Shell-Skript *klebe3* würde das gleiche leisten wie das zuvor vorgestellte Skript *klebe*:

```
$ cat klebe3
for datei
do
    file $datei | grep "text" >/dev/null 2>/dev/null # nur fuer grep
    if [ $? -ne 0 ]
    then
        echo "$datei keine Textdatei" >/dev/tty # nur fuer echo-Kommando
        continue
    fi
    echo "@@---$datei---@@"
    cat $datei
done >> zusamm # Alle Ausgaben der for-Schleife an zusamm anhaengen
$
```

3. Die gesamte Ausgabe eines »Programmiersprach-Kommandos« (**if**, **case**, **while**, **until** oder **for**) kann über eine Pipe in die Standardeingabe eines anderen Kommandos hineingeleitet werden. Ebenso kann die gesamte Eingabe für ein »Programmiersprach-Kommando« aus einer Pipe gelesen werden.

Beispiel

Das nachfolgende Shell-Skript *langnam* gibt alle Dateinamen eines Directories (ist als Argument anzugeben) aus, die länger als 8 Zeichen sind:

```
$ cat langnam
if [ $# -ne 1 ]
then
    echo "usage: $0 directory"
    exit 1
fi

for i in `ls -a $1`
do
    echo "$i"
done |
while read name
do
    laenge=`expr "$name" : '.*'`
    if [ $laenge -gt 8 ]
```

```
        then
            echo "$name ($laenge)"
        fi
    done
$ chmod u+x langnam
$ langnam .
addiere.c (9)
adresse.txt (11)
basisname1 (10)
basisname2 (10)
countdown2 (9)
countdown (9)
div_null.c (10)
div_null2 (9)
eing_zaehl (10)
gruessoft (9)
gruss_egon (10)
gruss_hans (10)
hallo.arc (9)
holedatei (9)
lottodruck (10)
mom_inhalt (10)
muenz_kom (9)
suchtext1 (9)
wartemil2 (9)
wd_inhalt (9)
zaehle.txt (10)
$
```

4. Normalerweise wird für die Kommandos **if**, **case**, **while**, **until** und **for** keine eigene Subshell gestartet. Somit sind die Werte von Variablen, die innerhalb eines solchen Kommandos gesetzt werden, auch außerhalb dieses Kommandos bekannt:

Beispiel

```
$ cat for4
for i in 1 2 3 4; do echo $i; done
echo "Wert von i ist: $i"
$ chmod u+x for4
$ for4
1
2
3
4
Wert von i ist: 4
$
```

Wird dagegen für eines dieser »Programmiersprach-Kommandos« Ein / Ausgabeumlenkung verwendet, werden diese Kommandos in einer eigenen Subshell ausgeführt. In diesem Fall sind Veränderungen von Variablen, die innerhalb eines solchen Kommandos vorgenommen wurden, außerhalb nicht verfügbar:

```
$ cat for5
for i in 1 2 3 4; do echo $i; done >/dev/tty
echo "Wert von i ist: $i"
$ chmod u+x for5
$ for5
1
2
3
4
Wert von i ist:
$
```

Ebenso wird eine eigene Subshell aufgerufen, wenn eines der »Programmiersprach-Kommandos« in einer Pipe-Konstruktion angegeben wird.

Beispiel

Das nachfolgende Shell-Skript *langzeil1* zählt alle Zeilen einer Datei (als erstes Argument anzugeben), die mehr als 40 Zeichen haben:

```
$ cat langzeil1
zaehl=0
cat $1 |
while read zeile
do
    laenge=`expr "$zeile" : '.*'`
    if [ $laenge -gt 40 ]
    then
        zaehl=`expr $zaehl + 1`
    fi
done
echo "$1 hat $zaehl Zeilen, die laenger als 40 Zeichen"
$ chmod u+x langzeil1
$ langzeil1 lottodruck
lottodruck hat 0 Zeilen, die laenger als 40 Zeichen
$
```

In diesem Fall würde die Verwendung von **export**, wie im nachfolgenden Shell-Skript *langzeil2* auch nicht weiterhelfen¹:

```
$ cat langzeil2
zaehl=0
export zaehl      # Nun wird zaehl exportiert
cat $1 |
while read zeile
do
    laenge=`expr "$zeile" : '.*'`
    if [ $laenge -gt 40 ]
    then
        zaehl=`expr $zaehl + 1`
    fi
done
```

1. Als Erinnerung: In Subshells vorgenommene Änderungen an exportierten Variablen haben keinen Einfluß auf die Werte dieser Variablen in der Elternshell.

```
    fi
done
echo "$1 hat $zaehl Zeilen, die laenger als 40 Zeichen"
$ chmod u+x langzeil2
$ langzeil2 lottodruck
lottodruck hat 0 Zeilen, die laenger als 40 Zeichen
$
```

Eine mögliche Lösung für dieses Problem ergäbe sich hier unter der Verwendung des builtin-Kommandos **exec**, um die Standardeingabe für die Ausführungsdauer der durch **while** bedingten Subshell in die Datei **\$1** umzulenken:

```
$ cat langzeil3
zaehl=0
exec < $1 # Standardeingabe nach $1 umlenken

while read zeile # nun keine Umlenkung notwendig
do
    laenge=`expr "$zeile" : '.*'`
    if [ $laenge -gt 40 ]
    then
        zaehl=`expr $zaehl + 1`
    fi
done

exec < /dev/tty # Standardeingabe wieder auf Terminal umlenken

echo "$1 hat $zaehl Zeilen, die laenger als 40 Zeichen"
$ chmod u+x langzeil3
$ langzeil3 lottodruck
lottodruck hat 2 Zeilen, die laenger als 40 Zeichen
$
```

4.12.7 Funktionen

Seit UNIX-System V.2 erlaubt die Bourne-Shell – wie höhere Programmiersprachen – die Definition von Funktionen. Die Syntax für eine Funktionsdefinition ist dabei:

```
funktionsname() { kdoliste; }
```

Zwischen { und dem ersten Kommando aus *kdoliste* muß mindestens ein Leerzeichen, Tabulatorzeichen oder Neuezeile-Zeichen angegeben sein. Zwischen dem letzten Kommando aus *kdoliste* und der abschließenden } muß immer ein Semikolon angegeben sein, wenn sich diese beiden in derselben Zeile befinden.

Damit wird der Funktion *funktionsname* die *kdoliste* zugeordnet.

Der Aufruf von *funktionsname* bewirkt dann die Ausführung der zugeordneten *kdoliste*.

Beispiel

Nachfolgend wird ein neues »Kommando« **ll** eingeführt, welches alle Dateien eines Directorys mit den Optionen **-CF** listet und danach die Anzahl der Dateien ausgibt. Der Nachteil ist hier, daß **ll** immer nur alle Dateien des Working-Directors auflistet:

```
$ ll() { ls -CF; echo "-----\n`ls | wc -l` Dateien"; }
$ ll
abc*          cpdir*          for3*          klebe3*        u*
abc**         cph*            for4*          kreiere*       umrech*
abc2*        cpzdr*          for5*          langnam*       userda*
addiere.c*   dateityp*        for5.aus       langzeil1*     uv*
adress*      datum*          gebdat*        langzeil2*     uvw*
adresse.txt* datv*            gls*           langzeil3*     uvwx*
ampell1*     del*            groesse*       letzilog*      var_wert*
ampel2*      demo*          gruesse*       ll*            vorh*
anfang*      diskuse*        gruesse2*      loesch*        wartemil*
argaus*      div_null*       gruessoft*     logfile*       wartemil2*
argu_pid*    div_null.c*     gruss*         lottodruck*    wartuser*
ausg*        div_null12*     gruss_egon*    lszaehl*       wd_inhalt*
ausgab*      dru*           gruss_hans*    mom_inhalt*    weiblich*
basisname1*  eing_zaehl*     hallo*         muenz_kom*     which1*
basisname2*  emilda*         hallo.arc*     namdatei*      which2*
catd*        emilda2*        hallo.c*       neudat*        woist*
cc2*         fahrsohn*       hexziff*       pd*            zaehle.txt*
cmo1*        fehler*         hexziff2*      sammel*        zdrucke*
cmo2*        fiba*           hier_dat*      schiebe*       zdruckel*
comp1*       fiba2*          holedat*       so**           zeichtyp*
copy1*       findch*         kinder1*       sortiert*      zeig_pid*
core*        finde_c*        kinder2*       stdio.h*       zusamm
countdow2*   for1*           klebe*         suchtext*
countdown*   for2*           klebe2*        suchtext1*
-----
118 Dateien
$
```

Da der Aufruf von Funktionen weitgehend dem Aufruf von Shell-Skripts entspricht, können auch bei Funktionsaufrufen Argumente angegeben werden, welche dann der entsprechenden Funktion in Form von Positionsparametern¹ zur Verfügung gestellt werden. Somit kann der Nachteil der vorherigen Funktion umgangen werden:

```
$ ll() { ls -CF "$@"; echo "-----\n`ls "$@" | wc -l` Dateien"; }
$ ll a*
abc*          addiere.c*   ampell1*      argaus*        ausgab*
abc**         adress*     ampel2*       argu_pid*
abc2*        adresse.txt* anfang*       ausg*
```

1. **\$0** enthält allerdings nicht den Funktionsnamen, sondern den Namen der ausführenden Shell.

```

    13 Dateien
$ ll /bin/c*
/bin/cat*      /bin/chmod*    /bin/conv*    /bin/cpio*
/bin/cc*       /bin/chown*    /bin/convert* /bin/cpr*
/bin/chgrp*    /bin/cmp*      /bin/cp*      /bin/csh*
-----
    12 Dateien
$

```

Eine Funktionsdefinition kann sich über mehrere Zeilen erstrecken. Die Shell gibt dann bei einer interaktiven Eingabe einer Funktionsdefinition solange den Sekundär-Promptstring aus, bis die Definition mit der abschließenden `}` beendet wird.

Beispiel

```

$ zaehle() {
> start=${1:-1}
> ende=${2:-10}
> step=${3:-1}
> i=$start
> while [ $i -le $ende ]
> do
> echo $i
> i=`expr $i + $step`
> done
> }
$ zaehle 2 35 7
2
9
16
23
30
$

```

Eigenschaften von Shell-Funktionen

Nachfolgend werden die Eigenschaften von Shell-Funktionen vorgestellt:

Für die Wahl eines Funktionsnamens gelten die gleichen Regeln wie für die Wahl von Datei- und Variablennamen¹. Gleiche Funktions- und Shellvariablen-Namen sind nicht erlaubt.

Beispiel

```

$ 2ma14() {
> echo `expr 2 \* 4`
> }
2ma14: is not an identifier
$ heute() { echo "Heute ist: `date +%d.%m.%y`" ; }
$ heute
Heute ist: 18.12.96
$ heute="Heute ist: `date +%d.%m.%y`"

```

1. Siehe Syntaxregeln für Bezeichner in Kapitel 3.

```
$ heute ↵
heute: not found      [heute keine Funktion mehr, sondern nun eine Variable]
$ echo $heute ↵
Heute ist: 18.12.96
$
```

Shell-Funktionen sind nur in der Shell bekannt, in der sie definiert wurden, denn sie können nicht wie Shellvariablen an Subshells exportiert werden.

Beispiel

```
$ cdir() { cd $1; PS1="`pwd`> "; } ↵
$ cdir /bin ↵
/bin> cdir ↵
/user1/egon> cdir shellueb ↵
/user1/egon/shellueb> cdir shellueb ↵
/user1/egon/shellueb> export cdire ↵
cannot export functions
/user1/egon/shellueb> sh ↵
$ cdire ↵
cdire: not found
$ exit ↵
/user1/egon/shellueb> PS1="$ " ↵
$
```

Shell-Funktionen werden sehr oft entweder in der Datei *.profile* oder in einer eigenen »Funktionsdefinitions-Datei« definiert. Wird dann eine solche »Funktionsdefinitions-Datei« in einer Subshell mit dem Punkt-Kommando¹ gelesen, dann stehen die darin angegebenen Shell-Funktionen auch dieser Subshell zur Verfügung. Wichtig ist, daß die in *datei* enthaltenen Funktionsdefinitionen von der aktuellen Shell (und nicht von einer Subshell) gelesen werden.

Definitionen von häufig benötigten Funktionen können in der Datei *.profile* angegeben werden. Dadurch stehen sie immer sofort nach dem Anmelden zur Verfügung. Eine andere typische Vorgehensweise ist, nützliche Funktionen in einer eigenen »Funktionsdefinitions-Datei« (wie z.B. *funkdef*) im Home-Directory unterzubringen. Das Lesen dieser Datei mit dem Punkt-Kommando, wie z.B.

. \$HOME/funkdef

stellt dann diese Funktionen in der aktuellen Shell zur Verfügung. Wird dieser Aufruf in *.profile* angegeben, so werden die entsprechenden Funktionen der Login-Shell zur Verfügung gestellt, ohne daß dadurch *.profile* durch die explizite Angabe von Funktionsdefinitionen aufgebläht wird.

1. Aufrufsyntax: *. datei*

Das Punkt-Kommando *.* hat zur Folge, daß die Shell die Kommandos in *datei* liest und ausführt; im Falle von Funktionsdefinitionen werden diese lediglich der aktuellen Shell zur Verfügung gestellt.

Die Angabe von Funktionsdefinitionen in einer eigenen Datei hat auch den Vorteil, daß in einer Subshell mit dem Aufruf

. \$HOME/funkdef

die in *funkdef* definierten Funktionen sehr schnell der Subshell zur Verfügung gestellt werden können, ohne daß die entsprechenden Funktionsdefinitionen nochmals zeitraubend und umständlich »von Hand« vollständig einzugeben sind.

Ein weiterer Vorteil solcher Funktionsdefinitions-Dateien ist, daß man mehrere Programme (Funktionen) in einer Datei unterbringen kann, was ja bei Shell-Skripts nicht möglich ist.

Im weiteren Verlauf sollen alle nützlichen Funktionen in der Datei *funkdef* gesammelt werden.

Beispiel

Als erstes werden die bereits vorgestellten Funktionen *ll*, *cdir* und *zaehle* in *funkdef* aufgenommen:

```
$ cd
$ pwd
/user1/egon
$ cat funkdef
#-- ll -----
#   ruft ls -CF auf und gibt Anzahl der aufgelisteten Dateinamen aus
ll() {
    ls -CF "$@"
    echo "-----\n`ls "$@" | wc -l` Dateien" # in Linux(bash): echo -e "..."
}

#-- cdir -----
#   entspricht dem Kommando cd, allerdings wird danach das
#   Working-Directory als Prompt-String ausgegeben
cdir() {
    cd $1
    PS1="`pwd`> "
}

#-- zaehle -----
#   realisiert eine for-Schleife mit step-Angabe
#       1.Argument: Startwert
#       2.Argument: Endwert
#       3.Argument: Schrittweite
zaehle() {
    start=${1:-1}
    ende=${2:-10}
    step=${3:-1}
    i=$start
```

```
while [ $i -le $ende ]
do
    echo $i
    i=`expr $i + $step`
done
}
$ cd shellueb
$ . $HOME/funkdef
$ cat sum_ungerad
. $HOME/funkdef
sum=0
for i in `zaehle 1 100 2`
do
    sum=`expr $sum + $i`
done
echo "Summe aller ungeraden Zahlen zwischen 1 und 100 ist: $sum"
$ chmod u+x sum_ungerad
$ sum_ungerad
Summe aller ungeraden Zahlen zwischen 1 und 100 ist: 2500
$
```

Ist eine Funktion erst einmal definiert, so läuft sie wesentlich schneller ab als ein Shell-Skript mit der gleichen Funktionalität. In diesem Fall entfällt nämlich das Lokalisieren der entsprechenden Skript-Datei innerhalb des Dateisystems und das nachfolgende Eröffnen dieser Skript-Datei.

Damit die Funktionsdefinitionen nach jedem Anmelden verfügbar sind, sollte nun am Ende von *.profile* folgende Zeile eingetragen werden:

```
. $HOME/funkdef
```

Beim Aufruf von Shell-Funktionen werden diese in der aktuellen Shell ausgeführt. Somit können sie verwendet werden, um Shell-Variablen in der momentan aktiven Shell zu verändern.

Beispiel

```
$ setze_name { name="$*"; }
$ name=franz
$ echo $name
franz
$ setze_name "Franz Mueller"
$ echo $name
Franz Mueller
$
```

Wäre `setze_name` dagegen ein Shell-Skript, so hätte die Zuweisung an `name` keine Auswirkung auf die aufrufende Shell:

```
$ cat setze_name
name="*"
$ chmod u+x setze_name
$ name=franz
$ echo $name
franz
$ ./setze_name "Franz Mueller" [relativer Pfadn., damit Shell-Skript]
$ echo $name [und nicht gleichnamige Funktion]
franz [aufgerufen wird]
$
```

Shell-Funktionen können auch rekursiv aufgerufen werden.

Beispiel

```
$ cat hexa
hex() {
    if [ -n "$1" ]
    then
        x=`expr $x \* 16`
        case "$1" in
            [aA]*) x=`expr $x + 10`;;
            [bB]*) x=`expr $x + 11`;;
            [cC]*) x=`expr $x + 12`;;
            [dD]*) x=`expr $x + 13`;;
            [eE]*) x=`expr $x + 14`;;
            [fF]*) x=`expr $x + 15`;;
            [0-9]*) x=`expr $x + `echo $1 | cut -c1``;;
            *) echo "nicht erlaubte Hexa-Ziffer: `echo $1 | cut -c1`" >&2
               exit 1;;
        esac
        hex `echo $1 | cut -c2-` # rekursiver Aufruf fuer restliche Ziffer
    fi
}

for i
do
    x=0
    hex $i
    echo "$i (16) = $x (10)"
done
$ chmod u+x hexa
$ hexa fff 12 ab
fff (16) = 4095 (10)
12 (16) = 18 (10)
ab (16) = 171 (10)
$ cp hexa ../bin
$ cat baum
baumchen() {
```

```
for i in `ls` # keine Dateinamen die mit Punkt beginnen
do
    if [ -d $i ]
    then
        echo $leer $i:
        leeralt=$leer
        cd $i
        leer="$leer---"; export leer
        baumchen $i
        leer=$leeralt
        cd ..
    else
        echo $leer $i
    fi
done
}

# main

dir=${1:-.}
cd $dir
baumchen

$ chmod u+x baum
$ baum /usr
adm:
--- pacct
--- pacct1
    :
    :
--- pacct8
--- pacct9
--- sa:
----- sa03
        :
        :
----- sa18
----- sar10
--- sulog
--- admin:
----- bupsched
----- checkfsys
----- gettyvalues
----- makefsys
----- menu:
----- DESC
----- diskmgmt:
----- DESC
----- checkfsys
----- cpdisk
----- erase
```

```
----- format
----- harddisk:
----- DESC
----- addbadblocks
----- addharddisk
----- checkhdfs
----- display
      :
      :
```

In den Anwendungsbeispielen zur Bourne-Shell (in Kapitel 4.22.5) wird eine verbesserte Version zur »baumartigen« Ausgabe von Directorybäumen gegeben.

Ein- und Ausgabeumlenkung ist sowohl bei der Definition als auch beim Aufruf einer Shell-Funktion erlaubt.

Beispiel

```
$ zahlausgab() {
> ende=${1:-20}
> i=1
> while [ $i -le $ende ]
> do
> if [ `expr $i % 2` -eq 0 ]
> then
> echo $i
> else
> echo $i >>ungerad
> fi
> i=`expr $i + 1`
> done
> echo "Fertig: Bis $ende gezaehlt" >&2
> }
$ zahlausgab > gerad 2>fehl
$ cat gerad
2
4
6
8
10
12
14
16
18
20

$ cat ungerad
1
3
5
7
9
```

```
11
13
15
17
19
$ cat fehl ↵
Fertig: Bis 20 gezaehlt
$
```

Die Definitionen von Shell-Funktionen gehören zum Environment eines Shell-Prozesses. Wird mit dem Systemaufruf **fork** ein Kindprozeß zur aktuellen Shell kreiert, erbt dieser das ganze Environment der aktuellen Shell und somit auch die Funktionsdefinitionen aus der Elternshell.

In Kapitel 4.4 wurde bereits erwähnt, daß zwei Möglichkeiten existieren, um ein Shell-Skript zu starten:

(1) **sh** *skriptname*

(2) *skriptname* (ohne **sh**)

Obwohl in beiden Fällen eine Subshell gestartet wird, bestehen doch bezüglich des Environments dieser Subshells gewisse Unterschiede.

Im Fall (1) wird das Shell-Programm **/bin/sh** gestartet und somit das Standard-Environment als Environment für die Subshell genommen, so daß in diesem Fall die in der Elternshell definierten Funktionen nicht zur Verfügung stehen.

Im Fall (2) dagegen wird das Environment der Elternshell für die Subshell genommen, so daß in diesem Fall alle in der Elternshell definierten Funktionen in der Subshell zur Verfügung stehen. Der Grund hierfür ist, daß bei (2) mit einem **fork**-Aufruf ein Kindprozeß (Subshell) erzeugt wird, so daß das Environment des Elternprozesses dupliziert wird. Das gleiche gilt auch für Subshells, die mit der **(..)**-Klammerung erzeugt werden.

Beispiel

```
$ . $HOME/funkdef ↵
$ cat sum_gerad ↵
sum=0
for i in `zaehle 0 100 2`
do
    sum=`expr $sum + $i`
done
echo "Summe aller geraden Zahlen zwischen 0 und 100 ist: $sum"
$ chmod u+x sum_gerad ↵
$ sum_gerad ↵
Summe aller geraden Zahlen zwischen 0 und 100 ist: 2550
$ sh sum_gerad ↵
sum_gerad: zaehle: not found
Summe aller geraden Zahlen zwischen 0 und 100 ist: 0
$ setze_name() { nam="$*"; } ↵
$ nam=franz ↵
```

```
$ echo $nam ↵
franz
$ (setze_name "Franz Mueller"; echo $nam) ↵ [Funktion bekannt]
Franz Mueller
$ echo $nam ↵
franz [die in Subshell gemachte Änderung an nam hat keinen Einfluß]
$ [auf nam in der Elternshell]
```

Builtin-Kommandos¹ können nicht durch eine Funktionsdefinition ersetzt werden, andere Kommandos dagegen sehr wohl.

Ob es sich bei einem Kommando um ein builtin-Kommando handelt oder nicht, kann sehr leicht mit dem builtin-Kommando **type** festgestellt werden. Die Aufrufsyntax für **type** ist:

type [kdo_name(n)]

Das Kommando **type** gibt aus, welches Programm ausgeführt wird, wenn einer der angegebenen *kdo_name(n)* als Kommando aufgerufen wird. **type** gibt somit für jeden einzelnen *kdo_namen* folgendes aus:

- ▶ entweder den absoluten Pfadnamen des entsprechenden Kommandos (eventuell mit dem Hinweis, daß dieses Kommando bereits in der internen **hash**-Tabelle² eingetragen wurde)
- ▶ oder die Information, daß das entsprechende Kommando ein builtin-Kommando ist
- ▶ oder die Information, daß das entsprechende »Kommando« eine Funktion ist; in diesem Fall wird noch die komplette Funktionsdefinition mit angezeigt.

Beispiel

```
$ type cd ↵
cd is a shell builtin
$ cd() { ↵
> cd $1 ↵
> PS1="`pwd`> " ↵
> } ↵
$ type cd ↵
cd is a shell builtin [Funktionsdefinition war umsonst]
$ type rm ↵
rm is /bin/rm [bzw.: rm is hashed (/bin/rm), wenn rm schon
aufgerufen wurde]

$ rm() { ↵
> case "$1" in ↵
> -f|-r|-i) /bin/rm $*;; ↵
> *) /bin/rm -i $*;; ↵
> esac ↵
```

1. Im Kapitel 4.15.8 sind alle builtin-Kommandos angegeben.
2. Siehe Kommando **hash** in Kapitel 4.15.5.

```
> } ↵
$ type rm ↵
rm is a function
rm(){
case "$1"-i | -r | -f)/bin/rm $* ;;*)/bin/rm -i $* ;;
}
$
```

Nach dieser Funktionsdefinition wird also immer beim Aufruf von **rm** diese Funktion und nicht das Kommando **rm** aufgerufen. Soll das Kommando **rm** aufgerufen werden, so ist der dazugehörige absolute Pfadname (**/bin/rm**) bzw. relative Pfadname anzugeben. Es ist im übrigen empfehlenswert, diese Funktionsdefinition für **rm**

```
:
:
#-- rm -----
#   loescht Dateien nicht sofort, sondern fragt nach
rm() {
  case "$1" in
    -f|-r|-i) /bin/rm $*;;
    *) /bin/rm -i $*;;
  esac
}
```

in *funkdef* mitaufzunehmen, um somit die Gefahr des zu voreiligen Löschens mit dem Kommando **rm** etwas zu entschärfen, da in diesem Fall vor jedem Löschen nochmals rückgefragt wird, ob wirklich gelöscht werden soll.

Die in der gerade aktiven Shell definierten Shell-Funktionen können mit dem Kommando

set (ohne Angabe von Argumenten)

angezeigt werden. Mit

unset funktionsname

kann die Definition einer Shell-Funktion wieder aufgehoben werden.

Der Aufruf des builtin-Kommandos

return [*n*]

bewirkt, daß eine Funktion mit dem Rückgabewert (exit-Status) *n* verlassen wird. Ist *n* nicht angegeben, dann wird als Rückgabewert der exit-Status des zuletzt in dieser Funktion ausgeführten Kommandos verwendet.

Beispiel

Es werden zwei Funktionen **pushd** und **popd** erstellt, die am Ende von **\$HOME/funkdef** anzufügen sind.

pushd wechselt dabei mit dem Aufruf der Funktion **cdir** zu dem als ersten Argument angegebenen Directory, so daß das neue Working-Directory immer im Prompt angezeigt wird. Zusätzlich speichert die Funktion **pushd** das vorherige Working-Directory, indem sie den Directory-Namen in der Variablen **PUSHD** aufhebt. Dort eventuell bereits vorhandene Directory-Namen werden dabei »nach rechts« geschoben. Die Variable **PUSHD** simuliert somit eine Art »Directory-Stack«, wobei die letzten Working-Directories immer an oberster Stelle im Stack stehen. Wird **pushd** ohne ein Argument aufgerufen, gibt es den momentanen Directory-Stack (Inhalt der Variablen **PUSHD**) aus.

Die Funktion **popd** ermöglicht nun – unter Verwendung dieses Directory-Stacks – eine sehr schnelle Rückkehr in vorherige Working-Directories. **popd** kann auf drei verschiedene Arten aufgerufen werden:

popd (ohne Argumente)	Rückkehr zum vorherigen Working-Directory
popd <i>n</i>	Rückkehr zum <i>n</i> -ten vorherigen Working-Directory
popd -	ganzer Directory-Stack wird entfernt; als Prompt wird wieder der default-Promptstring verwendet.

Solange die Funktionen **pushd** und **popd** zum Directory-Wechsel verwendet werden, sollte nicht das Kommando **cd**, sondern die Funktion **cdir** benutzt werden, da sonst der Prompt nicht dem wirklichen Working-Directory entspricht.

```
$ pwd
/user1/egon/shellueb
$ cat $HOME/funkdef
:
:
#-- pushd -----
#  wechselt - wie cdir - zum angegebenen Directory.
#  das aktuelle Directory wird allerdings in der Variablen
#  PUSHD festgehalten, so dass mit popd immer zum vorherigen
#  Directory zurueckgekehrt werden kann.
#  Wird kein Directory beim Aufruf angegeben, so gibt pushd
#  den gemerkten "Directory-Stack" aus.
pushd() {
    if [ $# -gt 1 ]
    then
        echo "usage: pushd [directory]" >&2
        return 1
    elif [ $# -eq 0 ]
    then
        echo "$PUSHD"
        return 0
    fi
}
```

```
if [ -z "$tiefe" ]
then
    tiefe=0
fi
if [ -d "$1" -a -x "$1" ]
then
    PUSHHD=`pwd` $PUSHHD
    tiefe=`expr $tiefe + 1`
    cd "$1"
else
    echo "Kann nicht zu $1 wechseln" >&2
    return 2
fi
}

#-- popd -----
#   ist das Gegenstueck zu pushd: Die als erstes Argument
#   angegebene Zahl legt fest, um wieviele Directories auf
#   dem "Directory-Stack" zurueckgekehrt werden soll.
#   Wird keine Zahl angegeben, so wird zur obersten Directory
#   des "Directory-Stacks" zurueckgekehrt.
#   Wird popd - aufgerufen, so wird der ganze "Directory-Stack"
#   geleert und wieder der default-Prompt eingestellt
popd() {
    if [ "$1" = "-" ]
    then
        unset PUSHHD tiefe
        PS1="$ "
        return 0
    fi
    zahl=${1:-1}
    if [ "$tiefe" -lt "$zahl" -o "$tiefe" -lt 1 ]
    then
        echo "zu kleiner Directory-Stack" >&2
        return 1
    fi
    d=`echo "$PUSHHD" | cut -d" " -f${zahl} -`
    cd "$d"
    s=`expr $zahl + 1`
    PUSHHD=`echo "$PUSHHD" | cut -d" " -f${s} -`
    tiefe=`expr $tiefe - $zahl`
}

$ . $HOME/funkdef
$ pushd /usr
/usr> pushd bin
/usr/bin> pushd ../include
/usr/include> pushd /etc
/etc> pushd
/usr/include /usr/bin /usr /user1/egon/shellueb [akt.Directory-Stack]
/etc> popd 2
```

```

/usr/bin> popd
/usr> cdir $HOME/bin           [wird nicht im Directory-Stack festgehalten]
/user1/egon/bin> pushd         [akt. Directory-Stack]
/user1/egon/shellueb
/user1/egon/bin> popd
/user1/egon/shellueb> popd
zu kleiner Directory-Stack
/user1/egon/shellueb> popd -
$ pwd
/user1/egon/shellueb
$

```

Beim Aufruf von Funktionen werden diese immer als solche erkannt, selbst wenn Quoting verwendet wird.

Beispiel

```

$ \c\d\i\r /usr
/usr> "cdir" include
/usr/include> 'cdir' ../bin
/usr/bin> \c'dir'
/user1/egon> "cdir shellueb" ["cdir shellueb" wird als Funktions- bzw.]
cdir shellueb: not found     [Kommandoname interpretiert (existiert
                             nicht)]
/user1/egon> cdi"r" shellueb
/user1/egon/shellueb> PS1="$ "
$

```

Als Erinnerung: Die Sonderbedeutung der Schlüsselwörter

```

if      then    else    elif    fi
case    esac
for     while   until   do      done
{       }

```

kann dagegen sehr wohl ausgeschaltet werden, indem für mindestens ein Zeichen dieser Wörter Quoting vorgenommen wird.

4.13 Fehlersuche in Shell-Skripts

Das Auffinden von syntaktischen und semantischen Fehlern in Shell-Skripts kann ein äußerst mühsames Unterfangen sein. Um die Fehlersuche etwas zu erleichtern, bietet die Bourne-Shell folgende Optionen an:

- n Kommandos werden nur gelesen und auf Syntaxfehler untersucht, aber nicht ausgeführt (*no execution*).
- v Alle Shell-Eingabezeilen werden so ausgegeben, wie sie gelesen werden. Erst nach dieser Ausgabe wird das entsprechende Kommando ausgeführt (*verbose*).

- x Alle Shell-Eingabezeilen werden auf die Standardfehlerausgabe ausgegeben, bevor sie ausgeführt werden; d.h., daß bei dieser Ausgabe bereits die Parametersubstitution, die Kommandosubstitution und die Dateinamen-expansion stattgefunden haben (*execution trace*).
- u Ein Zugriff auf nicht gesetzte Variablen bewirkt einen Fehler. Ohne diese Option wird üblicherweise die leere Zeichenkette geliefert (*unset*).
- e Das entsprechende Shell-Skript wird sofort verlassen, wenn ein Kommando einen exit-Status verschieden von 0 (nicht erfolgreich) liefert (*exit*).

Diese Optionen können auf zwei Arten verwendet werden:

```
sh [-nvxue] skript
```

In diesem Fall gelten die gesetzten Optionen nur für die Dauer der Skript-Ausführung.

```
set [-nvxue]
```

Hierbei bleiben die entsprechenden Optionen solange gesetzt, bis sie explizit mit + wieder ausgeschaltet werden:

```
set [+nvxue]
```

Im Unterschied zur vorherigen Aufrufform werden dabei die Optionen nicht für ein aufgerufenes Shell-Skript, das ja von einer eigenen Subshell ausgeführt wird, gesetzt, sondern gelten nur für die gerade aktive Shell. Um sie für ein Shell-Skript einzuschalten, müßten die betreffenden Optionen innerhalb des Shell-Skripts mit **set ...** gesetzt werden.

Beispiel

```
$ pwd
/user1/egon/shellueb
$ cat woist
dir=${2:-$HOME}
find $dir -name $1 -print
$ sh -x woist fiba
dir=/user1/egon
+ find /user1/egon -name fiba -print
/user1/egon/shellueb/fiba
$ set -x
$ pwd
+ pwd
/user1/egon/shellueb
$ woist zdrucke
+ woist zdrucke      [keine Ausg. der einz. Kdos in zdrucke, sondern nur
                     dessen Aufruf]
/user1/egon/shellueb/zdrucke
/user1/egon/bin/zdrucke
$ ls a[a-m]*
+ ls abc abc* abc2 addiere.c adress adresse.txt ampell ampel2
abc
abc*
abc2
```

```
addiere.c
adress
adresse.txt
ampel1
ampel2
$ set +x
+ set +x
$ ls a[a-m]*
abc
abc*
abc2
addiere.c
adress
adresse.txt
ampel1
ampel2
$
```

Die beiden meistbenutzten Optionen sind **-n** und **-x**. Die Option **-n** wird oft verwendet, um eine vorläufige Syntaxüberprüfung für ein Shell-Skript durchführen zu lassen, ohne daß dieses Shell-Skript gestartet werden muß. Dies hat natürlich seine Vorteile, da die Shell als Interpreter arbeitet und eventuelle Fehler erst an späterer Stelle in einem Skript auftreten könnten, nachdem schon ein Teil des Shell-Skripts ausgeführt wurde; die dabei durchgeführten Aktionen könnten dann nicht mehr rückgängig gemacht werden.

Doch die wohl am häufigsten benutzte Option ist **-x**. Sie eignet sich hervorragend dazu, ein Shell-Skript auszutesten, da sie bewirkt, daß die wirkliche Kommandozeile unmittelbar vor ihrer Ausführung – nachdem Parametersubstitution, Kommandosubstitution und Dateinamenexpandierung stattgefunden haben – angezeigt wird. Dies ist auch der Grund dafür, daß im nächsten Beispiel, welches das schrittweise Austesten eines Shell-Skripts zeigt, gerade von dieser Option ausgiebig Gebrauch gemacht wird.

Beispiel

Das nachfolgende Shell-Skript *werist* ermöglicht es, Informationen zu einzelnen Benutzern abzufragen: Login-Name, wirklicher Name, Home-Directory und Login-Shell. Die erste Version dieses Shell-Skripts enthält sowohl syntaktische als auch semantische Fehler, die dann schrittweise lokalisiert und behoben werden sollen. Für dieses Beispiel wird angenommen, daß die Datei */etc/passwd* den nachfolgend gezeigten Inhalt hat:

```
$ cat /etc/passwd
danko:9eY.3lQ6kjTQM:115:1:Daniel Korbmeier:/user1/danko:
bermu:qmZQCAXwDvCdk:116:1:Bernd Mueller:/user1/bermu:/bin/ksh
dani:VaIGULd4ze4c:117:1:David Niedermeier:/user1/dani:/bin/csh
kille:ltvvsLn4TdLDk:118:1:Kilian Lehner:/user1/kille:/bin/sh
$ nl -ba werist
1 while true
2 do
```

```

3      echo "
4      -----
5          Login-Name:          L
6          richtiger Name:      R
7
8          Programmende:        Q
9      -----
10
11      Gib ein: "
12      read antw
13      case $antw in
14          [qQ]) exit 0;;
15          [lLrR]) break;;
16          *) echo "\007Nur die Eingaben  L, R und Q erlaubt\n"
17              eco "Wiederholen Sie bitte Ihre Eingabe";;
18      esac
19  done
20
21  case $antw in
22      [lL]) echo "Zu suchender Login-Name (reg. Ausdruck moeglich) ?"
23              read such;;
24      [rR]) echo "Zu suchender richtiger Name (reg. Ausdruck moeglich) ?"
25              read such;;
26  esac
27
28  cat /etc/passwd |
29      while read zeile
30      do
31          gef=`echo $zeile | egrep $such`
32          if [ $gef != "" ]
33          then
34              loginnam=`echo $zeile | cut -f1 -d":"`
35              richtnam=`echo $zeile | cut -f5 -d":"`
36              homedir=`echo $zeile | cut -f6 -d":"`
37              loginshell=`echo $zeile | cut -f7 -d":"`
38              echo "-----"
39              echo " Login-Name: $loginnam; richtiger Name: $richtnam"
40              echo " Home-Directory: $homedir; Login-Shell: $loginshell"
41          fi
42      done
$ chmod u+x werist
$ werist

-----
Login-Name:          L
richtiger Name:      R

Programmende:        Q
-----

```

```
Gib ein:
1|
Zu suchender Login-Name (reg. Ausdruck moeglich) ?
dani
werist: test: argument expected    [Fehler !!]
$
```

Zum Auffinden dieses Fehlers eignet sich nun die Option **-x**:

```
$ sh -x werist
+ true
+ echo
-----
      Login-Name:      L
      richtiger Name:  R

      Programmende:    Q
-----

Gib ein:
-----
      Login-Name:      L
      richtiger Name:  R

      Programmende:    Q
-----

Gib ein:
+ read antw
1|
+ break
+ echo Zu suchender Login-Name (reg. Ausdruck moeglich) ?
Zu suchender Login-Name (reg. Ausdruck moeglich) ?
+ read such
dani
+ cat /etc/passwd
+ read zeile
+ echo danko:9eY.3lQ6kJTQM:115:1:Daniel Korbmeier:/user1/danko:
+ egrep dani
gef=
+ [ != ]                                [<---- Fehler liegt hier !!]
werist: test: argument expected
$
```

Die durch die Option **-x** bewirkte Ausgabe von Kommandozeilen wird mit einem vorangestellten Pluszeichen **+** (außer bei Variablen-Zuweisungen) gekennzeichnet.

Hier ist gut zu erkennen, daß der Fehler durch die leere Variable *gef* bedingt ist, da in diesem Fall die 32. Zeile von *werist*

```
if [ $gef != "" ]
```

zu folgender Zeile wird:

```
if [ != ]
```

Der Operator `!=` verlangt jedoch links und rechts einen Operanden. Deswegen sollte man die 32. Zeile in *werist* durch folgende Zeile ersetzen:

```
if [ -n "$gef" ]
```

Nach dieser Änderung kann nun das Shell-Skript wieder gestartet werden:

```
$ werist
```

```
-----
Login-Name:      L
richtiger Name:  R

Programmende:    Q
-----
```

Gib ein:

```
1
```

Zu suchender Login-Name (reg. Ausdruck moeglich) ?

```
ani
```

```
-----

Login-Name: danko; richtiger Name: Daniel Korbmeier  [<-- sollte nicht
                                                    angezeigt werden]
Home-Directory: /user1/danko; Login-Shell:
-----
Login-Name: dani; richtiger Name: David Niedermeier
Home-Directory: /user1/dani; Login-Shell: /bin/csh
$
```

Die falsche Ausgabe ist dadurch bedingt, daß **egrep** nicht nur im Feld des Login-Namens, sondern in der ganzen »passwd-Zeile« sucht. Der Fehler kann behoben werden, wenn die 31. Zeile in *werist*:

```
gef=`echo $zeile | egrep $such`
```

durch

```
gef=`echo $zeile | cut -f$feld -d":" | egrep $such`
```

ersetzt wird. Allerdings muß dann die Variable *feld* zuvor entsprechend gesetzt werden, so daß sich dann folgendes neue Listing für *werist* ergibt:


```
$ nl -ba werist
1 while true
2 do
3     echo "
4     -----
5         Login-Name:      L
6         richtiger Name:  R
7
8         Programmende:    Q
9     -----
10
11     Gib ein: "
12     read antw
13     case $antw in
14         [qQ]) exit 0;;
15         [lLrR]) break;;
16         *) echo "\007Nur die Eingaben L, R und Q erlaubt\n"
17             eco "Wiederholen Sie bitte Ihre Eingabe";;
18     esac
19 done
20
21 case $antw in
22     [lL]) echo "Zu suchender Login-Name (reg. Ausdruck moeglich) ?"
23         read such
24         feld=1;;
25     [rR]) echo "Zu suchender richtiger Name (reg.Ausdruck moeglich) ?"
26         read such
27         feld=5;;
28 esac
29
30 cat /etc/passwd |
31 while read zeile
32 do
33     gef=`echo $zeile | cut -f$feld -d":" | egrep $such`
34     if [ -n "$gef" ]
35     then
36         loginnam=`echo $zeile | cut -f1 -d":"`
37         richtnam=`echo $zeile | cut -f5 -d":"`
38         homedir=`echo $zeile | cut -f6 -d":"`
39         loginshell=`echo $zeile | cut -f7 -d":"`
40         echo "-----"
41         echo "  Login-Name: $loginnam; richtiger Name: $richtnam"
42         echo "  Home-Directory: $homedir; Login-Shell: $loginshell"
43     fi
44 done
$ werist
-----
Login-Name:      L
richtiger Name:  R
```

```

        Programmende:      Q
        -----

        Gib ein:
1 | 
Zu suchender Login-Name (reg. Ausdruck moeglich) ?
ani 
-----
        Login-Name: dani; richtiger Name: David Niedermeier
        Home-Directory: /user1/dani; Login-Shell: /bin/csh
$
```

Dieser Fehler ist nun behoben. Nun soll getestet werden, wie sich das Shell-Skript bei der Eingabe eines regulären Ausdrucks als Suchbegriff verhält:

```
$ werist 
-----
        Login-Name:      L
        richtiger Name:   R

        Programmende:    Q
        -----

        Gib ein:
1 | 
Zu suchender Login-Name (reg. Ausdruck moeglich) ?
da* 
$
```

Das scheint noch nicht zu funktionieren. Es soll nun wieder die Option **-x** verwendet werden, um den Fehler zu lokalisieren:

```
$ sh -x werist 
+ true
+ echo
-----
        Login-Name:      L
        richtiger Name:   R

        Programmende:    Q
        -----

        Gib ein:
-----
        Login-Name:      L
        richtiger Name:   R

        Programmende:    Q
        -----
```

```
Gib ein:
1 [↵]
+ read antw
+ break
+ echo Zu suchender Login-Name (reg. Ausdruck moeglich) ?
Zu suchender Login-Name (reg. Ausdruck moeglich) ?
+ read such
da* [↵]
feld=1
+ cat /etc/passwd
+ read zeile
+ echo danko:9eY.3lQ6kJTQM:115:1:Daniel Korbmeier:/user1/danko:
+ cut -f1 -d:
+ egrep dateityp datum datv          [<---- Fehler liegt hier]
gef=
+ [ -n ]
+ read zeile
+ echo bermu:qmZQCAXwDvCdk:116:1:Bernd Mueller:/user1/bermu:/bin/ksh
+ cut -f1 -d:
+ egrep dateityp datum datv          [<---- Fehler liegt hier]
gef=
+ [ -n ]
+ read zeile
+ echo dani:VasIGULd4ze4c:117:1:David Niedermeier:/user1/dani:/bin/csh
+ cut -f1 -d:
+ egrep dateityp datum datv          [<---- Fehler liegt hier]
gef=
+ [ -n ]
+ read zeile
+ echo kille:ltvvsLn4TdLDk:118:1:Kilian Lehner:/user1/kille:/bin/sh
+ cut -f1 -d:
+ egrep dateityp datum datv          [<---- Fehler liegt hier]
gef=
+ [ -n ]
+ read zeile
$
```

An dieser Ausgabe ist zu erkennen, daß der eingegebene reguläre Ausdruck nicht an **egrep** übergeben wird. Die Shell führt bereits die Dateinamenexpansion durch. Dieses Fehlverhalten kann behoben werden, indem die Variable `$such`, die den regulären Ausdruck enthält, bei der Übergabe an **egrep** in der 33. Zeile mit `".."` geklammert wird, um die Interpretation durch die Shell zu unterbinden:

```
gef=`echo $zeile | cut -f$feld -d":" | egrep "$such`
```

Nach dieser Änderung soll nun nochmals ein Versuch unternommen werden:

\$ **werist** 

```
-----  
Login-Name:      L  
richtiger Name:  R  
  
Programmende:    Q  
-----
```

Gib ein:

1 

Zu suchender Login-Name (reg. Ausdruck moeglich) ?

da* 

```
-----  
Login-Name: danko; richtiger Name: Daniel Korbmeier  
Home-Directory: /user1/danko; Login-Shell:  
-----
```

```
Login-Name: dani; richtiger Name: David Niedermeier  
Home-Directory: /user1/dani; Login-Shell: /bin/csh
```

\$

Diese Ausgabe enthält noch ein kleines Manko: Wenn nämlich kein Eintrag im letzten Feld einer *passwd*-Zeile vorhanden ist¹, wird auch nichts für die Login-Shell ausgegeben. Besser wäre in diesem Fall */bin/sh* auszugeben. Dies kann erreicht werden, wenn nach der 39. Zeile in *werist* folgende Zeile eingefügt wird:

```
loginshell=${loginshell:-"/bin/sh"}
```

Nach dieser Änderung wird das Shell-Skript nochmals mit den gleichen Eingaben getestet:

\$ **werist** 

```
-----  
Login-Name:      L  
richtiger Name:  R  
  
Programmende:    Q  
-----
```

Gib ein:

1 

Zu suchender Login-Name (reg. Ausdruck moeglich) ?

da* 

```
-----  
Login-Name: danko; richtiger Name: Daniel Korbmeier  
Home-Directory: /user1/danko; Login-Shell: /bin/sh
```

1. Login-Shell ist diesem Fall die Bourne-Shell.

```
-----  
Login-Name: dani; richtiger Name: David Niedermeier  
Home-Directory: /user1/dani; Login-Shell: /bin/csh  
$
```

Nun scheint das Shell-Skript *werist* richtig zu sein, oder nicht ?

```
$ werist
```

```
-----  
Login-Name:      L  
richtiger Name:  R  
  
Programmende:    Q  
-----
```

Gib ein:

```
w
```

```
werist: eco: not found          [<---- Noch ein Fehler]
```

Nur die Eingaben L, R und Q erlaubt

```
-----  
Login-Name:      L  
richtiger Name:  R  
  
Programmende:    Q  
-----
```

Gib ein:

```
l
```

Zu suchender Login-Name (reg. Ausdruck moeglich) ?

```
bermu
```

```
-----  
Login-Name: bermu; richtiger Name: Bernd Mueller  
Home-Directory: /user1/bermu; Login-Shell: /bin/ksh  
$
```

Ist nun hoffentlich der letzte Fehler in Zeile 17 ausgebessert – das Ersetzen von **eco** durch **echo** – ergibt sich folgendes Aussehen für das Shell-Skript *werist*:

```
$ cat werist
```

```
while true
```

```
do
```

```
    echo "
```

```
-----  
Login-Name:      L  
richtiger Name:  R
```

```

        Programmende:      Q
        -----

        Gib ein: "
read antw
case $antw in
    [qQ])  exit 0;;
    [lLrR]) break;;
    *)    echo "\007Nur die Eingaben L, R und Q erlaubt\n"
          echo "Wiederholen Sie bitte Ihre Eingabe";;
esac
done

case $antw in
    [lL]) echo "Zu suchender Login-Name (reg. Ausdruck moeglich) ?"
          read such
          feld=1;;
    [rR]) echo "Zu suchender richtiger Name (reg. Ausdruck moeglich) ?"
          read such
          feld=5;;
esac
cat /etc/passwd |
while read zeile
do
    gef=`echo $zeile | cut -f$feld -d":" | egrep "$such"`
    if [ -n "$gef" ]
    then
        loginnam=`echo $zeile | cut -f1 -d":"`
        richtnam=`echo $zeile | cut -f5 -d":"`
        homedir=`echo $zeile | cut -f6 -d":"`
        loginshell=`echo $zeile | cut -f7 -d":"`
        loginshell=${loginshell:-"/bin/sh"}
        echo "-----"
        echo "  Login-Name: $loginnam; richtiger Name: $richtnam"
        echo "  Home-Directory: $homedir; Login-Shell: $loginshell"
    fi
done
$ cp werist ../bin
$
```

An diesem Beispiel ist zu erkennen, daß die Shell sehr wohl Unterstützung bei der Fehlersuche gibt, wobei diese Hilfestellungen natürlich nicht mit dem Komfort und den Möglichkeiten heutiger Debugger verglichen werden können.

4.14 Signalbehandlung in der Shell

Signale wurden bereits im ersten Buch dieser Reihe besprochen. Dort wurde die Signalbehandlung innerhalb von C-Programmen vorgestellt. Hier wird die Signalbehandlung in der Shell bzw. innerhalb von Shell-Skripts vorgestellt.

4.14.1 Allgemeines zu Signalen

Über ein Signal kann einem Prozeß eine bestimmte Botschaft geschickt werden. Die Ursachen für das Auftreten von Signalen können interner oder externer Natur sein.

Interne Signale treten auf, wenn während der Ausführung eines Prozesses (hier: des Shell-Prozesses) etwas Unplanmäßiges auftritt, wie

1. Ein aufgerufenes Kommando existiert nicht oder ist nicht ausführbar.
2. Fehler bei der Ein-/Ausgabeumlenkung (wie z.B. Eingabeumlenkung auf eine nicht existierende Datei).
3. Ein aufgerufenes Kommando beendet sich abnormal (wie z.B. bei Division durch 0).
4. Ein aufgerufenes Kommando beendet sich korrekt, aber mit einem exit-Status verschieden von 0.
5. Syntaxfehler bei den Programmiersprach-Konstrukten (wie z.B. **while** ohne Angabe von **do**).
6. Fehler beim Aufruf eines builtin-Kommandos (wie z.B. **cd** mit nicht existierenden Pfadnamen).

Die voreingestellte Reaktion der Shell auf diese Fehler ist:

- 1–3. Entsprechende Fehlermeldung wird ausgegeben und mit der Bearbeitung des nächsten Kommandos wird fortgefahren.
4. Mit der Bearbeitung des nächsten Kommandos wird ohne Fehlermeldung fortgefahren.
- 5–6. Hier ist zu unterscheiden zwischen einem Shell-Skript und einer interaktiven Shell. Eine Shell ist dann interaktiv, wenn sie entweder mit der Option **-i** aufgerufen wurde oder aber ihre Standardeingabe, Standardausgabe und Standardfehlerausgabe auf die Dialogstation eingestellt ist: Bei einer interaktiven Shell wird hier eine entsprechende Fehlermeldung ausgegeben und mit der Bearbeitung des nächsten Kommandos fortgefahren. Tritt einer dieser Fehler in einem Shell-Skript auf, so wird dessen Ausführung beendet.

Ist bei der Ausführung eines Shell-Skripts die Option **-e** gesetzt, so wird dieses Skript immer dann beendet, wenn die Ausführung eines Kommandos einen exit-Status verschieden von 0 liefert.

Externe Signale werden einem Prozeß (Shell-Prozeß) entweder direkt vom Benutzer (wie z.B. beim Drücken der *DEL*- oder *BREAK*-Taste) oder von einem anderen Prozeß (mit der Systemfunktion **kill** bzw. mit dem Kommando **kill**) geschickt.

4.14.2 Signalnummern

Die möglichen Signale sind durch ganzzahlige Nummern gekennzeichnet:

Signalnummer	Beschreibung
0	<i>terminate</i> : wird beim Verlassen einer Shell erzeugt.
1	<i>hangup</i> : wird beim Beenden einer Verbindung (z.B. Auflegen des Telefonhörers) erzeugt.
2	<i>intr</i> : Interrupt-Signal, welches durch Drücken der <i>DEL</i> - oder <i>BREAK</i> -Taste erzeugt wird.
3*	<i>quit</i> : wird durch Eingabe der Tastenkombination <i>Strg-\</i> erzeugt.
4*	<i>illegal instruction (not reset when caught)</i> : wird beim Versuch, einen illegalen Maschinenbefehl auszuführen, erzeugt.
5*	<i>trace trap (not reset when caught)</i> : wird erzeugt, wenn beim Debuggen eines Programms auf einen Haltepunkt (<i>Breakpoint</i>) aufgelaufen wird.
6*	<i>abort</i> : wird von der Systemfunktion <i>abort</i> erzeugt.
7*	<i>EMT-instruction</i> : wird bei der Ausführung eines EMT-Befehls (auf manchen Maschinen ohne Gleitpunkt-Rechnung) erzeugt.
8*	<i>floating point exception</i> : wird beim Auftreten eines Gleitpunktfehlers erzeugt (z.B. Division durch 0).
9	<i>kill (cannot be caught or ignored)</i> : bewirkt die sofortige Beendigung eines Prozesses und kann nicht abgefangen werden; wird z.B. beim Aufruf kill -9 pid an den Prozeß mit der Prozeßnummer <i>pid</i> geschickt.
10*	<i>bus error</i> : wird bei einem Fehler auf dem Systembus erzeugt (z.B. bei einem unzulässigen Zugriff über einen Zeiger in C).
11*	<i>segmentation violation</i> : wird beim Zugriff auf unerlaubte Adressen erzeugt (z.B. bei Zugriff über ungültige C-Zeiger).
12*	<i>bad argument to system call</i> : wird bei Übergabe eines unerlaubten Arguments an einen Systemaufruf erzeugt.

Tabelle 4.8: Mögliche Signalnummern

Signalnummer	Beschreibung
13	<i>write on a pipe with no one to read it</i> : wird beim Schreibversuch in eine gebrochene Pipe erzeugt; dies tritt z.B. auf, wenn ein Leseprozeß aus einer Pipe vorzeitig endet und der Schreibprozeß weiter in die gebrochene Pipe schreibt.
14	<i>alarm clock</i> : wird nach Ablauf der durch einen alarm -Aufruf vorgegebenen Zeitdauer erzeugt.
15	<i>software termination signal from kill</i> : voreingestelltes Signal beim Aufruf des Kommandos kill .
16	<i>user defined signal 1</i> : freies Signal, welches vom Benutzer definiert werden kann.
17	<i>user defined signal 2</i> : freies Signal, welches vom Benutzer definiert werden kann.

Tabelle 4.8: Mögliche Signalnummern (Fortsetzung)

Die mit * gekennzeichneten Signale bewirken – wenn sie nicht explizit abgefangen werden – nicht nur die Beendigung des jeweiligen Prozesses, sondern zusätzlich noch einen Speicherabzug; ein solcher Speicherabzug wird auch *core image* genannt und wird in die Datei *core* des Working-Directors geschrieben.

Allgemein gilt, daß das Eintreffen eines der obigen Signale zum Abbruch des betreffenden Prozesses führt, wenn dieser nicht explizite Vorkehrungen getroffen hat, um ein eingetroffenes Signal abzufangen.

Ein Prozeß kann dabei folgende Vorkehrungen zum Abfangen von Signalen¹ treffen:

- ▶ Signal ignorieren,
- ▶ Aufruf von Kommandos zur Signalbehandlung oder
- ▶ wieder die von der Shell voreingestellte Signalbehandlung herstellen.

4.14.3 Signale mit trap abfangen

Das Abfangen von Signalen in einer Shell, insbesondere innerhalb von Shell-Skripts, ist dabei mit dem Kommando **trap** möglich:

trap [argument] [signalnummer(n)]

Das Kommando **trap** legt die Reaktion der Shell auf asynchron eintreffende Signale fest. Trifft ein Signal ein, dessen Signalnummer in der Liste *signalnummer(n)* angegeben ist, dann führt die Shell die als *argument* angegebenen Kommandos aus.

1. Bis auf das Signal 9 können alle Signale abgefangen werden.

Danach setzt sie ihre Ausführung an der Stelle fort¹, an der die durch das Signal bedingte Unterbrechung stattfand.

Als Signalbehandlung ist dabei möglich:

1. Es kann für *argument* eine Liste von Kommandos angegeben werden ('*kdoliste*'), die bei Eintreffen eines der mit *signalnummer(n)* spezifizierten Signale auszuführen ist.
2. Ignorieren der Signale *signalnummer(n)*, indem für *argument* eine leere Zeichenkette (z.B. "" oder "") angegeben wird.
3. Wird beim Aufruf von **trap** kein *argument* angegeben, so wird für die Signale *signalnummer(n)* wieder die vom System voreingestellte Signalbehandlung festgelegt.

Werden beim Aufruf von **trap** kein *argument* und keine *signalnummer(n)* angegeben, dann gibt **trap** die Signalnummern aus, für die momentan mit Hilfe eines **trap**-Kommandos eine benutzerspezifische Signalbehandlung eingestellt wurde; zudem gibt es in diesem Fall zu jeder dieser Signalnummern die Kommandos an, die diese Signalbehandlung durchführen.

Beispiel

1. Es ist ein Shell-Skript *brkzaehl* zu erstellen, das sich beim Drücken der **BREAK**-oder **DEL**-Taste (Signal 2) nicht abbrechen läßt, sondern lediglich ausgibt, wie oft bereits diese Taste gedrückt wurde. Dieses Shell-Skript muß dann mit dem Signal 3 (Taste **Strg-**) abgebrochen werden:

```
$ cat brkzaehl ↵
# Signalbehandlung fuer intr (Signal 2)
trap ' i='expr $i + 1`; echo "$i.mal die DEL-Taste gedrueckt" ' 2

# Signalbehandlung fuer quit (Signal 3)
trap ' echo "$0 wird nun abgebrochen"; exit 1 ' 3

i=0
while true
do
    sleep 1
done
$ chmod u+x brkzaehl ↵
$ brkzaehl ↵
[DEL]1.mal die DEL-Taste gedrueckt
[DEL]2.mal die DEL-Taste gedrueckt
[DEL]3.mal die DEL-Taste gedrueckt
[DEL]4.mal die DEL-Taste gedrueckt
[DEL]5.mal die DEL-Taste gedrueckt
[Strg-\]brkzaehl: 2056 Quit - core dumped
brkzaehl wird nun abgebrochen
$
```

1. Wenn bei den für *argument* angegebenen Kommandos nicht **exit** vorkommt.

2. Es kommt häufig vor, daß man seinen Bildschirmplatz wegen einer Besprechung verlassen muß. Für diese Zeit möchte man sich nicht unbedingt abmelden. Allerdings sind dann der Bildschirm und damit alle persönlichen Daten für jedermann zugänglich. In solchen Fällen ist es nützlich, ein Shell-Skript *bildsperr* zu besitzen, das den Bildschirm sperrt und nur nach Eingabe eines entsprechenden Paßwortes eine Weiterarbeit zuläßt. Allerdings wäre ein solches Shell-Skript nur wenig wert, wenn es einem fremden Benutzer möglich wäre, dieses Shell-Skript mit der *BREAK*-Taste abzubrechen:

```
$ pwd
/user1/egon/shellueb
$ cat bildsperr
# Festlegen des Passworts
# -----
tput clear # Bildschirm loeschen

while true
do
    stty -echo # echo-Fkt. ausschalten für verdeckte Eingabe
    echo "Bitte Passwort eingeben: \c"
    read passwd1
    echo "\nPasswort-Eingabe wiederholen: \c"
    read passwd2
    if [ "$passwd1" != "$passwd2" ]
    then
        echo "\n\007Verschiedene Passwort-Eingaben"
        echo "Passwort-Eingabe muss wiederholt werden\n\n"
    else
        break
    fi
done

# Bildschirm bis zur Eingabe des richtigen Passworts sperren
# -----
trap '' 1 2 3 15 # Ab jetzt die Signale 1 2 3 15 ignorieren
while true
do
    tput clear # Bildschirm loeschen
    echo "Passwort: \c"
    read passwd
    echo
    if [ "$passwd" != "$passwd1" ]
    then
        echo "\007Falsches Passwort"
        sleep 2
    else
        stty echo # echo-Funktion wieder einschalten
        break
    fi
```

```
done
$ chmod u+x bildsperr ↵
$ cp bildsperr ../bin ↵
$
```

3. Die bei **trap** als *argument* angegebene Kommandoliste wird zweimal gelesen: das erstemal bei der Ausführung des **trap**-Kommandos, das zweitemal, wenn die Kommandoliste, bedingt durch das Eintreffen eines Signals, aufgerufen wird. Deshalb empfiehlt es sich, die angegebene Kommandoliste mit `'.'` zu klammern, um Parametersubstitution, Kommandosubstitution oder Dateinamenexpansion beim erstmaligen Lesen auszuschalten:

```
$ cat fang1 ↵
x=$HOME/bin
trap 'ls $x' 2 [<---- Für $x wird bereits hier /user1/egon/bin eingesetzt]
x="a*"
while true
do
    sleep 1
done
$ chmod u+x fang1 ↵
$ fang1 ↵
[DEL]bildsperr
copy
cpdir
cpzdr
del
gebdat
gruesse
hexa
holedatei
letztlog
suchtext
wartuser
werist
zdrucke
[Strg-\]fang1: 2075 Quit - core dumped
$ cat fang2 ↵
x=$HOME/bin
trap 'ls $x' 2          [<---- $x hier noch nicht substit., sondern erst
                        beim Aufruf (a*)]
x="a*"
while true
do
    sleep 1
done
$ chmod u+x fang2 ↵
$ fang2 ↵
[DEL]abc
abc*
```

```
abc2
addiere.c
adress
adresse.txt
ampell
ampel2
anfang
argaus
argu_pid
ausg
ausgab
[Strg-\]fang2: 2837 Quit - core dumped
$
```

4. Bei im Hintergrund gestarteten Programmen werden zwar die Signale 2 (*intr*) und 3 (*quit*) ignoriert, aber nicht die Signale 1 (*hangup*) und 0 (*terminate*; Verlassen der Shell). Deshalb kann **trap** manchmal auch in einer Shell verwendet werden, wenn man das Kommando **nohup** nicht verwenden möchte:

```
$ pwd
/user1/egon/shellueb
$ ( trap '' 0 1; find / -name werist -print >gefunden 2>/dev/null ) &
$ exit
```

In diesem Fall wird das **find**-Kommando durch das Abmelden nicht abgebrochen. Beim nächsten Anmelden sollte der Pfadname von *werist* in der Datei */user1/egon/shellueb/gefunden* stehen.

5. Sehr oft wird **trap** verwendet, um im Falle des Abbruchs eines Shell-Skripts noch Aufräumarbeiten (*cleanup*) durchzuführen, wie z.B. das Entfernen von temporären Dateien. Das nachfolgende Shell-Skript *psuser* listet alle Prozesse der Benutzer, die auf der Kommandozeile beim Aufruf von *psuser* angegeben sind:

```
$ cat psuser
tmp_name=/tmp/psuser.$$
trap 'rm -f $tmp_name 2>/dev/null; exit 1' 0 1 2 3 15 # cleanup

ps -ef >$tmp_name

cat $tmp_name | line
echo "-----"
for user
do
    fgrep "$user" $tmp_name
    echo "-----"
done
$ chmod u+x psuser
```

```
$ psuser egon root
  UID  PID  PPID  C   STIME TTY      TIME COMMAND
-----
  egon  667    1   1 12:18:45 ttyic    0:03 -sh
  egon  722   667   4 12:23:30 ttyic    0:00 -sh
  egon  723   722  14 12:23:30 ttyic    0:00 ps -ef
-----
  root    0    0   0 Jan 11 ?        0:00 sched
  root    1    0   0 Jan 11 ?        0:10 /etc/init
  root    2    0   0 Jan 11 ?        0:00 vhand
  root    3    0   0 Jan 11 ?        0:07 bdflush
  root  122    1   0 10:49:54 console 0:01 /etc/getty console console
  root   74    1   0 10:49:43 ?        0:05 /etc/cron
  root  124    1   0 10:49:57 vt01     0:01 /etc/getty /dev/vt01 vt01
  root  125    1   0 10:49:57 vt02     0:01 /etc/getty /dev/vt02 vt02
  root  126    1   0 10:49:57 ttyia    0:00 /etc/getty ttyia 9600
  root  127    1   0 10:49:57 ttyib    0:00 /etc/getty ttyib 9600
  root  129    1   0 10:49:58 ttyid    0:00 /etc/getty ttyid 9600
  root  130    1   0 10:49:58 ttyie    0:00 /etc/getty ttyie 9600
  root  131    1   0 10:49:59 ttyif    0:01 /etc/getty ttyif 9600
-----
$ cp psuser ../bin
$
```

6. Wird **trap** ohne Argumente aufgerufen, so werden alle Signalnummern ausgegeben, für die momentan eine benutzerspezifische Signalbehandlung eingestellt ist; zu jedem abzufangenden Signal werden dabei die Kommandos angegeben, die diese Signalbehandlung durchführen:

```
$ trap 'echo "Ade, $LOGNAME"' 0
$ trap
0: echo "Ade, $LOGNAME"
$ exit
Ade, egon
```

7. Werden die folgenden Zeilen in der Datei *.profile* eingetragen, dann würde bei jedem neuen Anmelden automatisch in das Directory gewechselt, welches das Working-Directory zum Zeitpunkt des letzten Abmeldens war. Zudem würde noch der Zeitpunkt der letzten Abmeldung ausgegeben:

```
trap 'echo "`pwd` `date`" >${HOME}/.zuletzt' 0
if [ -f ${HOME}/.zuletzt ]
then
    read workdir datum <${HOME}/.zuletzt
    cd $workdir
    echo "\n**** Letzte Abmeldung war: $datum ****"
fi
```

Hinweise

- ▶ Die Liste der möglichen Signalnummern ist systemabhängig und kann üblicherweise in der Datei `/usr/include/sys/signal.h` »nachgeschlagen« werden.
- ▶ Ist ein leeres Argument bei **trap** angegeben, so werden die entsprechenden Signale ignoriert:

```
trap '' 0 1 2      # Signale 0, 1 und 2 ignorieren
```

Wird ein Signal ignoriert, so ignorieren auch alle Subshells dieses Signal. Wird jedoch eine bestimmte Signalbehandlung (nicht leere Kommandoliste) für ein Signal festgelegt, so wird diese Signalbehandlung nicht an Subshells weiter vererbt; diese behalten dann weiter die voreingestellte Signal-Behandlung:

```
$ trap 'echo "DEL-Taste gedrueckt"' 2   
$ trap '' 3  [Ignorieren des Signals 3; wird vererbt]  
$ trap ';' 15  [Nicht-Ignorieren des Signals 15; wird nicht  
vererbt]  
  
$ trap   
0: echo "`pwd` `date`" > $HOME/.zuletzt  
2: echo "DEL-Taste gedrueckt"  
3:  
15: ;  
$
```

- ▶ Der Versuch, das Signal mit der Signalnummer 11 (*segmentation violation*) abzufangen, resultiert in einem Fehler.
- ▶ Die Signale 2 (*intr*) und 3 (*quit*) werden für ein Kommando, das im Hintergrund gestartet wird, ignoriert.

4.15 Builtin-Kommandos der Bourne-Shell

Die builtin-Kommandos sind Teile des Programms **sh**. Deswegen hat die Shell keinen neuen Prozeß zu starten, um diese ablaufen zu lassen. Deshalb starten diese Kommandos »schneller«¹ als die anderen (nicht builtin-) Kommandos. Seit System V Release 2 ist für die builtin-Kommandos auch Ein-/Ausgabeumlenkung möglich.

Während die meisten dieser Kommandos wichtige Werkzeuge für Shell-Skripts sind, ist es für andere wichtig, daß sie in der momentan aktiven Shell und auf keinen Fall in einer Subshell ablaufen².

Zunächst werden in diesem Kapitel alle noch nicht behandelten builtin-Kommandos ausführlich besprochen. Am Ende folgt eine Zusammenfassung aller builtin-Kommandos der Bourne-Shell.

1. Siehe Kapitel 4.16: Abarbeitung von Kommandozeilen.

2. Ein Kandidat für diese Gruppe ist z.B. das Kommando **cd**.

4.15.1 Das Null-Kommando (:)

Das Null-Kommando : liefert ohne jegliche weitere Aktionen den exit-Status 0 (erfolgreich). Seine Aufrufsyntax ist:

: [argument(e)]

Es unterscheidet sich in zwei wesentlichen Punkten vom Kommentar-Kommando #:

1. Obwohl das Null-Kommando selbst keine Aktion ausführt, so wird doch – anders als beim Kommentar-Kommando # – die angegebene Kommandozeile durch die Shell ausgewertet.

Beispiel

```
$ # `echo "Guten Morgen" > frueh`  
$ cat frueh  
cat: cannot open frueh  
$ : `echo "Guten Morgen" > frueh`  
$ cat frueh  
Guten Morgen  
$
```

Das Null-Kommando ignoriert zwar das durch die Auswertung der angegebenen Argumente gelieferte Ergebnis, kann aber nicht die Auswertung dieser Argumente durch die Shell unterbinden.

2. Beim Null-Kommando wird nicht bedingungslos – wie beim Kommentar-Kommando – der Rest der Kommandozeile ignoriert, sondern nur der Text bis zum nächsten Kommando. Und ein Kommando kann bekanntlich nicht nur mit dem Neuezeile-Zeichen, sondern auch mit einem Semikolon abgeschlossen werden.

Beispiel

```
$ # Ausgabe des Working-Directorys ; pwd  
$ : Ausgabe des Working-Directorys ; pwd  
/user1/egon/shellueb  
$
```

Sehr schön läßt sich auch der Unterschied zwischen den Kommandos # und : beim nachfolgenden Beispiel erkennen:

```
$ # Kommentar- \  
$ : Kommentar- \  
> zeile  
$
```

Während das Kommentar-Kommando # den Rest einer Zeile (einschließlich dem Fortsetzungszeichen) vollständig ignoriert, gilt für den Aufruf des Null-Kommandos das gleiche wie bei anderen UNIX-Kommandos, wo die Metazeichen der Shell ihre Sonderbedeutung behalten; somit bezieht es sich auch nicht bedingungslos auf eine Eingabezeile, sondern auf eine Kommandozeile.

Typische Anwendungen

- Das Null-Kommando : wird häufig verwendet, um einem Kommando einen Kommentar voranzustellen:

```
$ : Ausgabe des Login-Namens und des work. dir. ; logname; pwd ↵
egon
/user1/egon/shellueb
$
```

- Das Null-Kommando wird auch oft verwendet, um in Shell-Skripts eine Endlos-Schleife zu realisieren:

```
while :                # entspricht der Angabe: while true
do
    .....
    if bedingung
    then
        break
    fi
    .....
done
```

Da das Null-Kommando als exit-Wert immer 0 liefert, wird die **while**-Bedingung immer erfüllt sein; eine solche Endlos-Schleife wird üblicherweise beim Eintreten einer bestimmten Bedingung mit dem Kommando **break** abgebrochen.

- Mit dem Null-Kommando : kann ein leerer **then**-Zweig in einer **if**-Anweisung realisiert werden:

```
if [ $# -lt 2 ]
then
    :                # nach then muss immer mind. ein Kommando
                    # angegeben sein; sonst Syntaxfehler
else
    echo "Hoechstens 2 Argumente erlaubt" >&2
    echo "usage: ...." >&2
    exit 1
fi
```

4.15.2 Argumente als Kommandos ausführen (eval)

Das Kommando **eval** bewirkt, daß eine Kommandozeile zweimal von der Shell gelesen wird, bevor sie diese ausführt. Die Aufrufsyntax des **eval**-Kommandos ist:

```
eval [argument(e)]
```

Die Shell liest die bei **eval** angegebenen *argument(e)* und führt diese dann als Kommandos aus. Diese Vorgehensweise bringt allerdings mit sich, daß die angegebenen *argument(e)* zweimal ausgewertet¹ werden:

- ▶ das erstmal beim Lesen der **eval**-Kommandozeile durch die Shell und
- ▶ das zweitemal bei der Ausführung des **eval**-Kommandos, das die einmal ausgewerteten *argument(e)* als eigentliche Kommandozeile interpretiert.

Vergleichbar mit der { }-Klammerung werden die als *argument(e)* angegebenen Kommandos in der momentanen Shell und nicht in einer Subshell ausgeführt.

Beispiel

```
$ ls | wc -l ↵
135
$ eval ls | wc -l ↵
135 [Ausgabe wie oben, da Kdozeile zwar zweimal
    gelesen wird, aber immer die gleiche bleibt]
$ list=ls ↵
$ zaehle='| wc -l' ↵
$ $list $zaehle ↵
|: No such file or directory [Fehler, da nach einmalig. Lesen nur
wc: No such file or directory Param.subst. stattfand und somit
-1: No such file or directory der Inhalt von zaehle als Argumente an
                             ls uebergeben wird]
$ eval $list $zaehle ↵
135
$
```

Beim letzten Aufruf werden die beiden Shellvariablen *zaehle* und *list* zunächst ausgewertet, so daß folgende Aufrufzeile resultiert:

```
eval ls | wc -l
```

Da **eval** nun ein zweites Lesen der Kommandozeile erzwingt, wird nun die aus der Auswertung der Argumente resultierende Kommandozeile ausgeführt:

```
ls | wc -l
$ x=3; echo '$x' ↵
$x
$ x=3; eval echo '$x' ↵ [x=3; eval echo $x --> echo 3]
3
$
```

Das nachfolgende Shell-Skript *letztarg* gibt immer das letzte übergebene Argument aus:

```
$ cat letztarg ↵
eval echo $$# # \$ notw., um erstes $ ueber erste Auswertung zu retten
$ chmod u+x letztarg ↵
$ letztarg eins zwei drei vier ↵
vier
$ echo b* ↵
```

1. D.h., es wird zweimal Parametersubstitution, Kommandosubstitution und Dateinamenexpansion durchgeführt.

```

basisname1 basisname2 baum bildsperr brkzaehl
$ letztag b*
brkzaehl
$

```

Das nachfolgende Shell-Skript `lc` listet Dateinamen mit den Optionen **-CF** auf. Wird die Option **-p** beim Aufruf angegeben, so erfolgt die Ausgabe seitenweise (mit **pg**):

```

$ pwd
/user1/egon/shellueb
$ cat lc
# Dateien mit den Optionen -CF auflisten
#
# usage: lc [-p]
#          -p   Seitenweise Ausgabe
#
if [ "$1" = "-p" ]
then
    PG=" | /usr/bin/pg"
    shift
else
    PG=""
fi

eval /bin/ls -aCF "$@" $PG
$ chmod u+x lc
$ lc $HOME/bin
./          copy*      del*        hexa*       psuser*     werist*
../         cpdir*     gebdat*     holedat*    suchtext*   zdrukce*
bildsperr* cpzdr*     gruesse*    letztlog*   wartuser*
$ cp lc ../bin
$

```

Hinweis

Es dürfen auch mehrere **eval**-Kommandos hintereinander angegeben werden, was zu einer entsprechend geschachtelten Auswertung führt:

Beispiel

Eine Variable `u` enthält den Text `"$uv"`. Die Variable `uv` wiederum enthält den Text `"$uvw"`. Die Variable `uvw` enthält ihrerseits den Text `"$uvw"`. Die Variable `uvw` schließlich enthält den Text `"ls b*"`. Es soll nun mit einem Aufruf unter Verwendung von `$u` der `ls`-Befehl der Variablen `uvw` ausgeführt werden:

```

$ u='$uv'
$ uv='$uvw'
$ uvw='$uvw'
$ uvw='ls b*'
$ eval eval $u
basisname1
basisname2
baum

```

```
bildsperr  
brkzaehl  
$
```

Um die einzelnen Auswertungsschritte nachvollziehen zu können, wird die Option **-x** für die Ausführung dieser Kommandozeile gesetzt:

```
$ set -x  
$ eval eval eval $u  
+ eval eval eval $uv  
+ eval eval $uvw  
+ eval $uvw  
+ ls basisname1 basisname2 baum bildsperr brkzaehl  
basisname1  
basisname2  
baum  
bildsperr  
brkzaehl  
$ set +x  
+ set +x  
$
```

Bei einer Zuweisung eines Wertes an eine Shell-Variable wird von der Shell üblicherweise nur auf der rechten Seite des Zuweisungsoperators die Parametersubstitution durchgeführt:

```
$ a=x  
$ b=5  
$ $a=$b  
x=5: not found [x=5 wird von der Shell als Kommandoname interpretiert]  
$
```

Unter Verwendung von **eval** kann die Parametersubstitution auch für die linke Seite einer Zuweisung erreicht werden:

```
$ a=x  
$ b=5  
$ eval $a=$b  
$ echo $x  
5  
$
```

Prinzipiell kann also **eval** verwendet werden, um eine Art von Zeigervariablen in Shell-Skripts zu verwenden:

```
$ zahl1=1000  
$ zahl2=50  
$ zgr_zahl=zahl1  
$ eval echo \$$zgr_zahl  
1000  
$ zgr_zahl=zahl2
```

```
$ eval echo \$$zgr_zahl ↵  
50  
$
```

Typische Anwendung

eval wird meist in Shell-Skripts verwendet, die dynamisch während ihrer Ausführung Kommandozeilen aufbauen.

Beispiel

Es ist ein Shell-Skript *datlaufe* zu erstellen, das zu allen Dateien eines Directorybaums deren Namen ausgibt und dann ein Kommando einliest, das für diese Datei durchzuführen ist. Der Name des Directorybaums ist als Argument anzugeben. Wird kein Argument angegeben, so wird das Working-Directory angenommen:

```
$ cat datlaufe ↵  
laufe() {  
    for i in *  
    do  
        while echo "`pwd`/$i: \c"  
            read kdo  
        do  
            if [ -z "$kdo" ]  
            then  
                break  
            fi  
            eval $kdo $i  
        done  
        if [ -d $i ]  
        then  
            cd $i  
            laufe  
            cd ..  
        fi  
    done  
}  
  
dir=${1:-.}  
if [ ! -d "$dir" ]  
then  
    echo "$dir ist kein directory" >&2  
    echo "usage: $0 [directory]" >&2  
    exit 1  
fi  
cd $dir  
laufe  
$ chmod u+x datlaufe ↵  
$ datlaufe $HOME/bin ↵  
/user1/egon/bin/bildsperr: ↵  
/user1/egon/bin/copy: ↵  
/user1/egon/bin/cpdir: file ↵
```

```

cpdir:                                commands text
/user1/egon/bin/cpdir: 
/user1/egon/bin/cpzdrr: cat
until [ -z "$1" ]
do
    > $HOME/zeildruck/$1.syn    # Anlegen der Synchronisationsdatei
    cp $1 $HOME/zeildruck
    rm $HOME/zeildruck/$1.syn    # Entfernen der Synchronisationsdatei
    shift
done
/user1/egon/bin/cpzdrr: ls -l
-rwxr--r--  1 egon      other      196 Dec 13 14:16 cpzdrr
/user1/egon/bin/cpzdrr: 
/user1/egon/bin/del: type
del is /user1/egon/bin/del
/user1/egon/bin/del: 
/user1/egon/bin/gebdat: [DEL]
$ cp datlaufe ../bin
$

```

eval wird auch oft verwendet, um auf den Wert des letzten Positionsparameters zuzugreifen, z.B. mit

```
eval letztarg='${${#}}'
```

4.15.3 Überlagern der Shell mit einem Kommando (exec)

Das Kommando **exec** bewirkt, daß das Code-, Daten- und Stacksegment der aktuellen Shell durch ein anderes Programm ersetzt wird. Die Aufrufsyntax für **exec** ist:

```
exec [argument(e)]
```

Das über die angegebenen *argument(e)* festgelegte Kommando wird anstelle des Shellprogramms ausgeführt, ohne daß hierfür eine Subshell (Kindprozeß) kreiert wird. Die Shell wird somit durch das über die *argument(e)* spezifizierte Kommando überlagert, was dazu führt, daß mit Beendigung des Kommandos auch die aktuelle Shell beendet wird.

Beispiel

```
exec date
```

Die aktuelle Shell wird durch das Kommando **date** überlagert. Das heißt, daß nach der Ausgabe des Datums die aktuelle Shell beendet wird. Wenn es sich also bei der aktuellen Shell um die Login-Shell handelt, muß sich der Benutzer danach neu anmelden.

```
exec /bin/csh
```

Die aktuelle Shell wird durch die C-Shell (*/bin/csh*) überlagert. Das heißt, daß von nun ab in der C-Shell gearbeitet wird, die nicht als eigener Kindprozeß gestartet wird, sondern die neue interaktive Shell ist. Wenn also die C-Shell (mit **exit**)

beendet wird, ist damit auch diese UNIX-Sitzung beendet, falls es sich dabei um die Login-Shell handelte. Wird diese Zeile in die Datei *.profile* eingetragen, arbeitet der entsprechende Benutzer nach dem Anmelden immer in der C-Shell, obwohl die Bourne-Shell als Login-Shell für ihn in */etc/passwd* eingetragen ist.

Typische Anwendung

Sind beim **exec**-Aufruf keine *argument(e)* angegeben, sondern nur Umlenkungsanweisungen, werden diese für die aktuelle Shell ausgewertet:

```
$ exec >sh.out ↵
$ ls b* ↵
$ exec >/dev/tty ↵ [Standardausgabe wieder auf das Terminal legen]
$ cat sh.out ↵
basisname1
basisname2
baum
bildsperr
brkzaehl
$
```

Dies kann verwendet werden, um Ausgaben einer interaktiven Shell in einer Datei festzuhalten oder aber Eingaben an eine Shell aus einer Datei lesen zu lassen.

4.15.4 Auswerten der Kommandozeilen-Optionen eines Shell-Skripts (getopts)

Das Kommando **getopts** wird in Shell-Skripts verwendet, um Kommandozeilen zu lesen und die dort angegebenen Optionen auszuwerten. Die Aufrufsyntax für **getopts** ist:

```
getopts optstring name [argument(e)]1
```

Jedesmal, wenn **getopts** in einem Shell-Skript aufgerufen wird, dann liefert es die nächste Option aus der Kommandozeile und weist diese der Shell-Variablen *name* zu; zudem wird der Index (Positionsparameter-Nummer) des nächsten zu bearbeitenden Arguments in der Shell-Variablen **OPTARG** abgelegt. **OPTARG** ist beim Aufruf des entsprechenden Shell-Skripts zunächst immer auf 1 gesetzt.

Mit *optstring* werden die für das entsprechende Skript zugelassenen Optionen festgelegt. Wenn nach einer Option ein Argument oder eine Argumentengruppe² verlangt ist, sollte nach dieser Option ein Doppelpunkt in *optstring* angegeben werden: So würde z.B. die Angabe **ab:le** für *optstring* bedeuten, daß **-a**, **-b**, **-l** und **-e** gültige Optionen sind, wobei hinter den beiden Optionen **-b** und **-e** ein weiteres Argument anzugeben ist.

1. Dieses Kommando ist neu ab System V.3.

2. Siehe Hinweise.

Wenn eine Option, die ein zusätzliches Argument erfordert, beim Durcharbeiten der Kommandozeile gefunden wird, dann wird das entsprechende Argument der Shell-Variablen **OPTARG** zugewiesen.

Wird eine ungültige Option in der Kommandozeile gefunden, dann wird der Shell-Variablen *name* das Zeichen **?** zugewiesen.

Wenn **getopts** keine weiteren Optionen in einer Kommandozeile findet oder das spezielle Argument **--**¹ liest, dann liefert es einen von 0 verschiedenen exit-Status. Diese Konvention bringt es mit sich, daß **getopts** sehr oft als **while**-Bedingung eingesetzt wird.

Normalerweise liest **getopts** den Inhalt der Positionsparameter **\$1**, **\$2**, ... eines Shell-Skripts. Wenn allerdings beim Aufruf von **getopts** zusätzliche *argument(e)* angegeben wurden, liest **getopts** diese anstelle der Positionsparameter.

Beispiel

1. Ein Shell-Skript sei für die Optionen **-a -b -l** ausgelegt, wobei hinter der Option **-b** ein weiteres Argument bzw. eine weitere Argumentengruppe anzugeben ist.

Der folgende Ausschnitt aus diesem Shell-Skript zeigt, wie diese Optionen abgearbeitet werden können:

```
.....
FEHLMELD="$0: -$option ist nicht erlaubt und wird ignoriert"
OPTIONA=0
OPTIONB=0
OPTIONL=0
while getopts ab:l option
do
    case $option in
        a) OPTIONA=1;;
        b) OPTIONB=1; ARGB=$OPTARG;;
        l) OPTIONL=1;;
        \?) echo $FEHLMELD
            exit 2;;
        esac
    done
    shift `expr $OPTIND - 1`
.....
```

Die **while**-Schleife bewirkt, daß **getopts** so lange Optionen verarbeitet, bis es einen exit-Status verschieden von 0 liefert. Dieser Fall tritt ein, wenn alle vorgegebenen Optionen verarbeitet sind oder wenn **--** gelesen wird.

Die **case**-Anweisung verwendet den Wert von *option*, um festzulegen, welche Anweisungen während jedes Schleifendurchlaufs durchgeführt werden

1. **--** kann in der Kommandozeile angegeben werden, um das Ende der angegebenen Optionen anzuzeigen.

müssen. Die Shell-Variable **OPTIND** wird jedesmal inkrementiert, wenn ein neuer Positionsparameter zur Abarbeitung anfällt. Sie zeigt an, wie viele von den vorgegebenen Optionen bereits gelesen wurden. Der Ausdruck

```
shift `expr $OPTIND - 1`
```

bewirkt, daß das erste Argument, das keine Option ist, nach **\$1**, das nächste nach **\$2** usw. geschoben wird.

Das obige Skript würde z.B. folgende Kommandozeilen akzeptieren:

```
skriptname -a -l -b "xxx y zz" ...
skriptname -a -l -b "xxx y zz" -- ...
skriptname -a1 -b xxx,y,zz ...
skriptname -a1 -b "xxx y zz" ...
skriptname -b xxx,y,zz -l -a ...
```

Natürlich sind auch Aufrufe erlaubt, bei denen keine oder nicht alle Optionen angegeben sind.

2. Das nachfolgende Shell-Skript **wer** ist eine etwas abgeänderte Version des Kommandos **who**. Es bietet folgende Optionen an:

-i Informationen zu Benutzer (Login-Name, richtiger Name, Home-Directory)

-m mail an alle angemeldeten Benutzer verschicken

-n nach Namen sortieren

-z nach Anmeldezeitpunkt sortieren

Die erste Version dieses Shell-Skripts **wer1** wird auf herkömmliche Weise (ohne Verwendung von **getopts**) erstellt:

```
$ cat wer1 
postsend() {
    tmp_name="/tmp/wer.$$"
    > $tmp_name      # Anlegen der leeren Datei $tmp_name
    echo "Welche Nachricht ist zu senden (Abschluss mit Strg-d):"
    cat </dev/tty >>$tmp_name    # Einlesen des mail-Textes
    who |
    eval $SORTIER |
    cut -d' ' -f1 |
    while read name
    do
        echo "\nmail an $name (j/n): \c"
        antw=`line </dev/tty`
        if [ "$antw" = "j" -o "$antw" = "J" ]
        then
            mail $name < $tmp_name
        fi
    done
```

```
        rm -f $tmp_name
    }

    ANZEIGE="normal"
    SORTIER="sort"

    for i
    do
        case $i in
            -i) ANZEIGE="info";;
            -m) ANZEIGE="post";;
            -n) SORTIER="sort";;
            -z) SORTIER="sort -b +2";;
            *) echo "$0: unerlaubte Option $i" >&2
               echo "usage: $0 [-i] [-m] [-n] [-z]" >&2
               echo "      -i   Infos aus /etc/passwd" >&2
               echo "      -m   mail an alle angemeldeten Benutzer" >&2
               echo "      -n   nach Namen sortieren" >&2
               echo "      -z   nach Anmeldezeitpunkt sortieren" >&2
               exit 1;;
        esac
    done

    case $ANZEIGE in
        normal) eval who | $SORTIER;;
        info) for nam in `who | $SORTIER | cut -d" " -f1`
              do
                  zeile=`grep "^$nam:" /etc/passwd`
                  loginnam=`echo $zeile | cut -d":" -f1`
                  richtnam=`echo $zeile | cut -d":" -f5`
                  homedir=`echo $zeile | cut -d":" -f6`
                  echo "$loginnam ($richtnam; $homedir)"
              done;;
        post) postsend;;
    esac
    $ chmod u+x werl
    $
```

Der Nachteil dieser Realisierung ist allerdings, daß alle Optionen einzelnen anzugeben sind, wie z.B.

```
werl -i -z
```

Eine Zusammenfassung der Optionen beim Aufruf von `wer1` wäre nicht möglich¹:

```
$ wer1 -iz
wer1: unerlaubte Option -iz
usage: wer1 [-i] [-m] [-n] [-z]
        -i   Infos aus /etc/passwd
        -m   mail an alle angemeldeten Benutzer
        -n   nach Namen sortieren
        -z   nach Anmeldezeitpunkt sortieren
$
```

Dieser Nachteil wird mit der Verwendung von **getopts** in der zweiten Version `wer` beseitigt:

```
$ cat wer
postsend() {
    :
    :
}

ANZEIGE="normal"
SORTIER="sort"

while getopts imnz opt
do
    case $opt in
        i) ANZEIGE="info";;
        m) ANZEIGE="post";;
        n) SORTIER="sort";;
        z) SORTIER="sort -b +2";;
        \?) echo "usage: $0 [-i] [-m] [-n] [-z]" >&2
            echo "        -i   Infos aus /etc/passwd" >&2
            echo "        -m   mail an alle angemeldeten Benutzer" >&2
            echo "        -n   nach Namen sortieren" >&2
            echo "        -z   nach Anmeldezeitpunkt sortieren" >&2
            exit 1;;
    esac
done

case $ANZEIGE in
    normal) eval who | $SORTIER;;
    info) for nam in `who | $SORTIER | cut -d" " -f1`
        do
            zeile=`grep "^$nam:" /etc/passwd`
            loginnam=`echo $zeile | cut -d":" -f1`
            richtnam=`echo $zeile | cut -d":" -f5`
            homedir=`echo $zeile | cut -d":" -f6`
```

1. Dies könnte zwar auch realisiert werden, wäre aber wesentlich umständlicher als bei **getopts**.

```

        echo "$loginam ($richtnam; $homedir)"
    done;;
post) postsend;;
esac
$ chmod u+x wer
$ cp wer ../bin
$

```

3. Das nachfolgende Shell-Skript *erinner* ermöglicht es, einen Text nach einer bestimmten Zeit an ein Terminal zu schicken. Voreinstellung ist das Terminal, an dem *erinner* aufgerufen wird. Ist ein anderes Terminal erwünscht, muß der Name dieses Terminals als letztes Argument angegeben werden. Der zu schickende Text kann dabei auf der Kommandozeile (**-t text**) oder interaktiv eingegeben werden. Zudem kann über die Optionen angegeben werden, ob der Text in Banner-Form (**-b**) zu schicken und nach wie vielen Minuten (**-m minuten**) er zu senden ist. Die Option **-h** ermöglicht die Ausgabe von usage-Information zu diesem Shell-Skript *erinner*:

```

$ cat erinnern
# erinnern Version v1.0 (7.1.1997)
#
# erinnert den Benutzer nach einer bestimmten Zeit
#
# Syntax: erinnern [-m minuten] [-t text] [-b] [-h] [Terminalname]
#
# -m minuten Erinnerung nach minuten (Voreinst.: 10 Min.)
# -t text Erinnerungstext (Voreinst.: Text einlesen)
# -b Erinnerungstext in Banner-Form ausgeben
# -h nur usage-Info ausgeben
# Terminalname Erinnerung an Terminalname ausgeben
# (Voreinst.: Terminal, an dem aufgerufen)
#
# Autor: Egon ...
#

usage() {
    echo "usage: $0 [-m minuten] [-t text] [-b] [-h] [Terminalname]
        -m minuten Erinnerung nach minuten (Voreinst.: 10 Min.)
        -t text Erinnerungstext (Voreinst.: Text einlesen)
        -b Erinnerungstext in Banner-Form ausgeben
        -h nur usage-Info ausgeben
        Terminalname Erinnerung an Terminalname ausgeben
        (Voreinst.: Terminal, an dem aufgerufen)" >&2
}

kdo="echo"

while getopts m:t:bh opt
do
    case $opt in
        m) MIN="$OPTARG";;
        t) TEXT="$OPTARG";;

```

```
        b) kdo="banner";;
        h) usage
           exit 0;;
        \?) usage
           exit 1;;
    esac
done
shift `expr $OPTIND - 1`

TTY=${1:-"tty"}
if [ ! -c "/dev/$TTY" ]
then
    echo "Fehler: Terminal \"/dev/$TTY\" existiert nicht" >&2
    exit 2
fi

MIN=${MIN:-"10"}
if [ -z "$TEXT" ]
then
    echo "Gib Erinnerungstext ein (max. eine Zeile) ein: \c" >&2
    read TEXT
fi

echo "Text \"$TEXT\" wird in $MIN Min. an Terminal /dev/$TTY geschickt" >&2

{ sleep `expr $MIN * 60`; eval $kdo "$TEXT" >/dev/$TTY; } &

$ chmod u+x erinnerer
$ erinnerer -b -t "Besprechung nicht vergessen" -b -m 5 tty12
Text "Besprechung nicht vergessen" wird in 5 Min. an Terminal /dev/tty12
geschickt
$ cp erinnerer ../bin
$
```

Hinweise

Mit System V Release 3 wurden gewisse Syntaxregeln aufgestellt, die alle neu hinzukommenden Kommandos einhalten müssen. Die von System V.3 vorgegebenen Regeln werden zwar nicht von allen zur Zeit verfügbaren UNIX-Kommandos eingehalten, aber alle neu hinzukommenden Kommandos werden diesen vorgegebenen Regeln folgen. Diese 13 Syntaxregeln¹ sind:

1. Kommandonamen müssen zwischen 2 und 9 Zeichen lang sein.
2. Kommandonamen dürfen nur Kleinbuchstaben² und Ziffern enthalten.
3. Optionen müssen immer genau ein Zeichen lang sein.
4. Allen Optionen muß ein - (Minuszeichen) vorangestellt werden.

1. Die Regeln 3-10 beziehen sich auf die Optionen.

2. Keine Umlaute oder ß.

5. Optionen ohne Argumente können hinter einem - (Minuszeichen) gruppiert angegeben werden.
6. Das erste Argument zu einer Option muß mit Leer- und / oder Tabulatorzeichen von der Option getrennt angegeben werden (z.B. ist **-o datei** erlaubt, aber nicht **-odatei**).
7. Argumente zu Optionen dürfen nicht optional sein.
8. Argumente in einer Argumentgruppe zu einer Option müssen entweder durch Kommata (z.B. **-o xxx,z,yy**) oder mit Leer- und / oder Tabulatorzeichen voneinander getrennt sein; in diesem Fall ist Quoting zu verwenden (z.B. **-o "xxx z yy"**)
9. Alle Optionen müssen vor eventuell anderen, auf der Kommandozeile vorhandenen Argumenten angegeben sein.
10. -- (doppeltes Minuszeichen) kann verwendet werden, um das Ende der Optionen-Angabe explizit anzuzeigen.
11. Der relativen Reihenfolge der Angabe von Optionen zueinander sollte keinerlei Bedeutung zugemessen werden.
12. Der relativen Reihenfolge der Angabe von anderen Argumenten kann sehr wohl vom jeweiligen Kommando eine Bedeutung zugemessen werden.
13. - (Minuszeichen) alleine mit voranstehenden bzw. nachfolgenden Leer- und / oder Tabulatorzeichen sollte nur verwendet werden, um damit die Standardeingabe zu spezifizieren.

Aus Konsistenzgründen sollte sich jeder Benutzer bei der Entwicklung eigener Kommandos oder Shell-Skripts an diese Konventionen halten.

- ▶ Das explizite Verändern der Shell-Variablen **OPTIND** kann zu unvorhersehbaren Ergebnissen führen.
- ▶ Das Kommando **getopts** ist ein neues Kommando seit System V Release 3. Frühere UNIX-Versionen bieten das vergleichbare Kommando **getopt** an, welches zwar auch noch in System V.3 verfügbar ist, aber heute nicht mehr verwendet wird.
- ▶ Der von der Shell-Variablen **OPTIND** bereitgestellte Index bezieht sich auf die Positionsparameter und nicht auf die Anzahl von bisher gelesenen Optionen.

Beispiel

- ```
(1) mll -a -l -x datei
(2) mll -al -x datei
```

Nachdem bei (1) die Option **-a** verarbeitet wurde, wird **OPTIND** auf 2 gesetzt, da sich die nächste Option **-l** in **\$2** befindet. Nachdem allerdings bei (2) **-a** verarbeitet wurde, wird **OPTIND** auf 1 gesetzt, da sich die nächste Option **-l** immer noch in **\$1** befindet.

### 4.15.5 Das hashing-Verfahren der Shell (hash)

Wird ein Kommando (nur unter Angabe des eigentlichen Kommandonamens<sup>1)</sup> aufgerufen, das kein builtin-Kommando und keine definierte Funktion der Shell ist, werden die in **PATH** angegebenen Directories benutzt, um nach diesem Kommando zu suchen. Um diesen zeitaufwendigen Suchvorgang bei einem erneuten Aufruf dieses Kommandos zu vermeiden, unterhält die Shell eine sogenannte hashing-Tabelle, in der für jedes aufgerufene Kommando, das kein builtin-Kommando und keine definierte Funktion ist, dessen Pfadname festgehalten wird. Der Inhalt dieser internen hashing-Tabelle kann nun mit dem Kommando **hash** abgefragt oder auch verändert werden. Die Aufrufsyntax des Kommandos **hash** ist:

**hash [-r] [name(n)]**

Wird **hash** ohne Argumente aufgerufen, so werden alle momentan gemerkten Kommandos aus der hashing-Tabelle mit zugehörigen Pfadnamen ausgegeben.

Bei der Ausgabe des **hash**-Kommandos werden drei Spalten ausgegeben:

- ▶ *hits* gibt an, wie oft ein entsprechendes Kommando aufgerufen wurde.
- ▶ *cost* ist ein Maß für den Aufwand, der notwendig war, um ein Kommando über die Suchpfade in **PATH** zu finden.
- ▶ *command* gibt den Pfadnamen des entsprechenden Kommandos an.

Falls *name(n)*<sup>2</sup> beim Aufruf von **hash** angegeben werden, dann werden diese, ohne daß sie ausgeführt werden, in dieser Tabelle eingetragen. Oft wird diese Aufrufform in der Datei *.profile* verwendet, um häufig verwendete Kommandos bereits beim Anmelden (über die **PATH**-Directories) suchen und in diese hashing-Tabelle eintragen zu lassen. So wird bei einem späteren Aufruf der entsprechenden Kommandos auch das erstmalige Suchen über die **PATH**-Variable unterbunden.

Wenn die Option **-r** (*remove*) angegeben wird, dann wird diese shell-interne Tabelle vollständig geleert. Dies ist besonders dann notwendig, wenn ein Kommando seine Position geändert hat, oder wenn ein erneutes Suchen erwünscht ist, z.B. wenn ein neues Kommando mit gleichem Namen in einem weiter vorne stehenden **PATH**-Directory eingerichtet wurde.

---

1. Ohne Pfadspezifikationen, d.h., ohne Verwendung des / bei Angabe des Kommandonamens.  
2. Namen von Kommandos.

```

Beispiel $ hash
hits cost command
3 2 /usr/bin/at
2 3 /user1/egon/bin/letzlog
1 2 /usr/bin/tput
1 3 /user1/egon/bin/gruesse
$ pwd
/user1/egon/shellueb
$ PATH=:/bin:/usr/bin
$ hash
hits cost command
$ wc -l gruesse
15 gruesse
$ cd ../bin
$ cat wc
echo "Ich bin ein Trojanisches Pferdchen"
echo "nun wird das wirkliche wc aufgerufen"
/bin/wc $@
$ chmod u+x wc
$ wc -l gruesse
15 gruesse
$ hash
hits cost command
1* 2 /bin/cat
1* 3 /bin/wc
$ hash -r
$ wc -l gruesse
15 gruesse
Ich bin ein Trojanisches Pferdchen
nun wird das wirkliche wc aufgerufen
15 gruesse
$ hash
hits cost command
1* 1 wc
$ rm -f wc
$ cd ../shellueb
$ wc -l gruesse
15 gruesse
$ hash
hits cost command
1* 3 /bin/wc
$

```

[Einträge sind durch die Aufrufe der entsprechenden]  
[Kommandos bzw. Shell-Skripts in .profile bedingt]

[Neues Setzen der PATH-Variablen]

[Neues Setzen von PATH bewirkte das Leeren der hash-Tabelle]

[wegen Eintrag in hash-Tabelle wird /bin/wc aufgerufen]

[Nach Löschen der hash-Tab. wird wc des Work.Dir. gefunden]

[kein wc in Work.Dir. bedingt neues Suchen und Finden von /bin/wc]



### 4.15.6 Setzen von Optionen für die aktuelle Shell (set)

Es ist möglich, während des Arbeitens in einer Shell, neue Optionen ein- bzw. auszuschalten; dazu steht das Kommando **set** zur Verfügung. Die Aufrufsyntax für **set** ist:

```
set [optionen [argument(e)]]
```

Das Kommando **set** kann recht vielseitig verwendet werden:

1. Wird **set** ohne Angabe von *optionen* oder *argument(e)* aufgerufen, dann zeigt es die Namen und Werte aller momentan definierten Shellvariablen und die Namen und Definitionen aller gerade definierten Funktionen an.
2. Wird **set** mit Angabe von *argument(e)* aufgerufen, so weist es diese Argumente<sup>1</sup> der Reihe nach den Positionsparametern **\$1**, **\$2** usw. zu.
3. **set** kann auch verwendet werden, um Optionen zu setzen oder wieder zu löschen, die das Verhalten der gerade aktiven Shell beeinflussen. Die unten genannten Optionen können auch nur für die Ausführungsdauer eines Shell-Skripts gesetzt werden; dazu müßten die gewünschten Optionen mit **set** innerhalb des Shell-Skripts gesetzt werden.

#### Optionen:

Bei den nachfolgend vorgestellten Optionen gilt:

- ▶ Vorangestelltes - (Minuszeichen) schaltet die entsprechende Option ein.
- ▶ Vorangestelltes + (Pluszeichen) schaltet die entsprechende Option aus.

**set** kennt nun folgende Optionen:

- a markiert die Shell-Variablen, die verändert oder neu angelegt werden, für den Export<sup>2</sup>.
- e (*exit*) bewirkt, daß ein Shell-Skript sofort beendet wird, wenn ein Kommando einen exit-Status verschieden von 0 (nicht erfolgreich) liefert.
- f (*filename expansion*) schaltet die Dateinamenexpandierung aus: Die Metazeichen \* ? [ werden als einfache Zeichen ohne Sonderbedeutung behandelt.
- h (*hash*) bewirkt, daß Kommandos innerhalb einer Funktion schon bei der Definition und nicht erst bei der Ausführung der Funktion lokalisiert und in der hashing-Tabelle<sup>3</sup> hinterlegt werden.

---

1. Angegebene Optionen zählen dabei nicht als Argumente.

2. Siehe builtin-Kommando **export** oder Kapitel 4.6.4 (Gültigkeitsbereiche von Variablen).

3. Siehe builtin-Kommando **hash** in Kapitel 4.15.5.

- k** (*keywords*) bewirkt, daß **alle** Shell-Variablen an die Umgebung (*environment*) eines Kommandos übergeben werden. Normalerweise werden Shell-Variablen nur dann an aufgerufene Kommandos übergeben, wenn sie beim Aufruf vor dem Kommandonamen angegeben sind<sup>1</sup>.
- n** (*no execution*) Kommandos werden nur gelesen und auf Syntaxfehler untersucht, aber nicht ausgeführt.
- t** (*terminate*) Nach dem Lesen und der Ausführung eines Kommandos wird die entsprechende Shell beendet.
- u** (*unset*) Ein Zugriff auf nicht gesetzte Shell-Variablen bewirkt einen Fehler. Ohne diese Option setzt die Shell üblicherweise die leere Zeichenkette für eine solche Variable ein.
- v** (*verbose*) bewirkt, daß alle Shell-Eingabezeilen – so wie sie gelesen werden – ausgegeben werden. Erst nach dieser Ausgabe wird das entsprechende Kommando ausgeführt.
- x** (*execute*) bewirkt, daß alle Kommandos und ihre Argumente – so wie sie wirklich von der Shell ausgeführt werden – ausgegeben werden; d.h., daß bei dieser Ausgabe bereits die Parametersubstitution, die Kommandosubstitution und die Dateinamenexpansion stattgefunden haben.
- (doppeltes Minuszeichen) hat keine Auswirkung auf die Optionen; ist nützlich, um **\$1** auf - zu setzen (**set --**).

Wie oben bereits erwähnt, kann eine Option ausgeschaltet werden, wenn anstelle eines Minuszeichens - ein Pluszeichen + vorangestellt wird.

Der automatische Parameter **\$-** enthält die gesetzten Optionen für die momentan aktive Shell. Mit dem Aufruf

```
echo $-
```

können diese gerade gesetzten Optionen ausgegeben werden.

**Beispiel**

```
$ ls abc*
abc
abc*
abc2
$ set -f
$ ls abc*
abc*
$ echo $-
sf
$ set +f
$ set *
$ echo $1
abc
```

[Option -s wird in Kapitel 4.17 vorgestellt]

---

1. Siehe Kapitel 4.6.4 (Gültigkeitsbereiche von Variablen).

```
$ set - *
$ echo $1
abc
$ ls xyz*
xyz: No such file or directory
$ set `ls xyz* 2>/dev/null`
:
:
[Ausgabe alle momentan definierten Shell-Variablen und
-Funktionen]
:
:
$ set - `ls xyz* 2>/dev/null` [Keine Ausgabe¹]
$ set - *
$ echo $1
abc
$
```

Der vorletzte Aufruf

```
set - *
```

zeigt, daß bei Angabe von - (Minuszeichen) der Positionsparameter **\$1** nicht auf -, sondern auf das erste »echte« Argument gesetzt wird; in diesem Fall ist dies der erste Dateiname im Working-Directory. Die Angabe von - stellt zusätzlich sicher, daß diese Zuweisung auch dann erfolgreich verläuft, wenn der erste Dateiname mit - beginnen würde.

### 4.15.7 Die builtin-Kommandos **readonly**, **times** und **ulimit**

Die drei letzten bisher noch nicht besprochenen builtin-Kommandos sind **readonly**, **times** und **ulimit**:

#### **readonly**

Das Kommando **readonly** ermöglicht es, Shell-Variablen als »nur lesbar« zu kennzeichnen. Die Aufrufsyntax von **readonly** ist:

```
readonly [variable(n)]
```

**readonly** markiert die angegebenen *variable(n)* als »nur lesbar«, was bedeutet, daß die Inhalte dieser *variable(n)* danach nicht mehr verändert werden können. Dieses Kommando kann also verwendet werden, um benannte Shell-Konstanten einzuführen. Wird **readonly** ohne Angabe von *variable(n)* aufgerufen, so werden alle Shell-Variablen ausgegeben, die momentan als »nur lesbar« markiert sind.

---

1. Diese Aufrufform kann in Anwendungsfällen wie z.B. beim Shell-Skript *zdrucke* in Kapitel 4.12.4 genutzt werden, wo ein Aufruf von **set** mit einer leeren Argumentenliste die Ausgabe aller definierten Variablen und Funktionen unterbinden soll.

Die angegebenen *variable(n)* können zum Zeitpunkt des Aufrufs von **readonly** bereits definiert sein oder erst später mit einer Zuweisung definiert werden.

### times

Das Kommando **times**<sup>1</sup> zeigt die bisher verbrauchte CPU-Zeit der momentan aktiven Shell an; es werden dabei zwei Zeiten ausgegeben:

1. Zeit (user)            gebrauchte CPU-Zeit im Benutzermodus
2. Zeit (sys)            gebrauchte CPU-Zeit im Systemmodus  
(z.B. bei der Ausführung von Systemroutinen)

#### Beispiel

```
$ times ↵
0m1s 0m8s
$ sh ↵
$ times ↵
0m0s 0m0s
$ exit ↵
$ { find / -name "*.c" -print | wc -l; times; } ↵
1263
0m1s 0m9s
$
```

### ulimit

Das Kommando **ulimit** legt ein Limit für die maximale Größe (*n* Blöcke) von Dateien fest, welche von der aktuellen Shell oder ihren Kindprozessen erstellt werden dürfen<sup>2</sup>. Die Aufrufsyntax für **ulimit** ist:

```
ulimit [n]
```

Wenn *n* nicht angegeben ist, so wird das momentan festgelegte Limit ausgegeben. Jeder Benutzer kann sein ihm zugestandenes Limit erniedrigen. Erhöhen dieses Limits ist allerdings nur dem Superuser vorbehalten.

## 4.15.8 Zusammenfassung der builtin-Kommandos

Hier werden nun alle builtin-Kommandos nochmals zusammengefaßt. Dabei wird jeweils eine kurze Beschreibung gegeben.

---

1. **times** verfügt über keinerlei Argumente.  
2. Für zu lesende Dateien gibt es keine Größenbegrenzung.

Die Bourne-Shell verfügt über folgende builtin-Kommandos:

| Kommando                                           | Wirkung                                                                                                                   |
|----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <b>#</b>                                           | Kommentar-Kommando                                                                                                        |
| <b>:</b> <i>[argument(e)]</i>                      | Null-Kommando                                                                                                             |
| <b>.</b> <i>kdodatei</i>                           | Punkt-Kommando<br><br>Liest die Kommandos in <i>kdodatei</i> und führt sie in der aktuellen Shell aus.                    |
| <b>break</b> <i>[n]</i>                            | Verlassen einer bzw. von <i>n</i> umschließenden <b>for-while-</b> bzw. <b>until-</b> Schleifen                           |
| <b>cd</b> <i>[directory]</i>                       | Wechseln in ein anderes Working-Directory                                                                                 |
| <b>continue</b> <i>[n]</i>                         | Abbrechen eines bzw. von <i>n</i> <b>for-, while-</b> bzw. <b>until-</b> Schleifendurchläufen                             |
| <b>echo</b> <i>[argument(e)]</i>                   | Ausgeben von Text                                                                                                         |
| <b>eval</b> <i>[argument(e)]</i>                   | Ausführen der <i>argument(e)</i> als Kommandos                                                                            |
| <b>exec</b> <i>[argument(e)]</i>                   | Überlagern der Shell mit einem Kommando                                                                                   |
| <b>exit</b> <i>[n]</i>                             | Beenden der momentan aktiven Shell                                                                                        |
| <b>export</b> <i>[variable(n)]</i>                 | Exportieren von Shell-Variablen                                                                                           |
| <b>getopts</b> <i>optstring name [argument(e)]</i> | Auswerten der Optionen in einem Shell-Skript                                                                              |
| <b>hash</b> <i>[-r] [name(n)]</i>                  | Abfragen bzw. Ändern der shell-internen hashing-Tabelle                                                                   |
| <b>newgrp</b> <i>[-] [gruppenname]</i>             | Kurzzeitiges Wechseln der Gruppenzugehörigkeit                                                                            |
| <b>pwd</b>                                         | Ausgeben des Working-Directorys                                                                                           |
| <b>read</b> <i>[variable(n)]</i>                   | Lesen einer Eingabezeile von der Standard-eingabe                                                                         |
| <b>readonly</b> <i>[variable(n)]</i>               | Shell-Variablen als »nur lesbar« kennzeichnen                                                                             |
| <b>return</b> <i>[n]</i>                           | Verlassen einer Funktion                                                                                                  |
| <b>set</b> <i> [--aefhknvux ] [argument(e)]</i>    | Setzen von Optionen für die aktuelle Shell bzw.<br><br>Zuweisen von Werten an die Positionsparameter                      |
| <b>shift</b> <i>[n]</i>                            | Verschieben der Werte von Positionsparametern                                                                             |
| <b>test</b> <i>[ausdr]</i>                         | Auswerten des Ausdrucks <i>ausdr</i> . Anstelle von <b>test</b> <i>ausdr</i> kann auch <b>[ ausdr ]</b> angegeben werden. |

| Kommando                                 | Wirkung                                                           |
|------------------------------------------|-------------------------------------------------------------------|
| <b>times</b>                             | Anzeigen der bisher von der aktuellen Shell verbrauchten CPU-Zeit |
| <b>trap</b> [argument] [signalnummer(n)] | Installieren bzw. Auflisten von Signalhandlern                    |
| <b>type</b> [kdo_name(n)]                | Anzeigen, welches Programm bei Kommando-Aufruf ausgeführt wird    |
| <b>ulimit</b> [n]                        | Festlegen einer maximalen Größe für Dateien                       |
| <b>umask</b> [3-stellige-oktalzahl]      | Setzen bzw. Ausgeben der Dateikreierungsmaske                     |
| <b>unset</b> [name(n)]                   | Löschen von Shell-Variablen oder Shell-Funktionen                 |
| <b>wait</b> [n]                          | Warten auf die Beendigung von Subshells (Kindprozessen)           |

## 4.16 Abarbeitung von Kommandozeilen

Die Shell arbeitet Kommandozeilen in folgender Reihenfolge ab:

1. Entfernen aller \Neuezeile-Zeichen.
2. Parametersubstitution und Auswerten von Variablenzuweisungen.
3. Kommandosubstitution.
4. Zerlegen der (aus den ersten 3 Schritten entstandenen) Kommandozeile in einzelne Worte.

Die Trennzeichen für die einzelnen Worte sind dabei in der Shell-Variablen **IFS** definiert. Während eine mit "" bzw. " geklammerte leere Zeichenkette als ein eigenes Wort interpretiert wird, gilt dies nicht für leere Zeichenketten, die ungeklammert sind.

### Beispiel

```
$ cat argausg ↵
echo "Die Parameterzahl ist $#"
```

```
echo "Inhalt von \$1 ist [$1]"
```

```
echo "Inhalt von \$2 ist [$2]"
```

```
echo "Inhalt von \$3 ist [$3]"
```

```
echo "Inhalt von \$4 ist [$4]"
```

```
$ chmod u+x argausg ↵
```

```
$ LEER= ↵
```

```
$ argausg "Leere Argumente" ' ' "" "$LEER" ↵
```

```
Die Parameterzahl ist 4
```

```
Inhalt von $1 ist [Leere Argumente]
```

```
Inhalt von $2 ist []
```

```
Inhalt von $3 ist []
```

```
Inhalt von $4 ist []
$ argausg "Leere Argumente" $LEER
Die Parameterzahl ist 1
Inhalt von $1 ist [Leere Argumente]
Inhalt von $2 ist []
Inhalt von $3 ist []
Inhalt von $4 ist []
$
```

5. Auswerten der Ein-/ Ausgabe-Umlenkungsangaben.
6. Expandierung von Dateinamen<sup>1</sup>.

Hierbei wird jedes einzelne Wort nach den Zeichen \* ? und [ durchsucht. Wenn eines dieser Zeichen in einem Wort gefunden wird, dann wird dieses Wort als *pattern* betrachtet, welches eine Vielzahl von Dateinamen abdecken kann. Jedes in der Kommandozeile gefundene pattern wird dann von der Shell expandiert, d.h. durch alle Dateinamen ersetzt<sup>2</sup>, die es abdeckt. Wenn kein Dateiname gefunden werden kann, den ein vorgegebenes pattern abdeckt, wird das entsprechende pattern nicht expandiert und bleibt somit unverändert. Zur Expandierung stehen folgende Metazeichen zur Verfügung:

| Metazeichen | Bedeutung                                         |
|-------------|---------------------------------------------------|
| *           | eine beliebige Zeichenfolge                       |
| ?           | ein beliebiges einzelnes Zeichen                  |
| [..]        | eines der in [..] angegebenen Zeichen             |
| [!..]       | ein Zeichen, welches nicht in [!..] angegeben ist |

7. Shell lokalisiert entsprechendes Kommando und führt es aus.

Die Lokalisierung des entsprechenden Kommandos lässt sich durch den nachfolgenden Programmablaufplan veranschaulichen.

1. Wurde ausführlich in Kapitel 4.7 beschrieben.  
2. Alphabetisch sortiert.

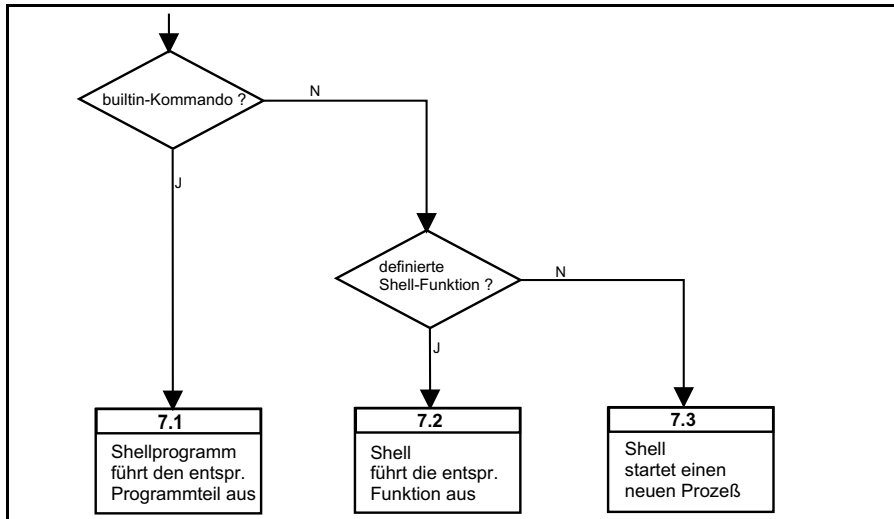


Abbildung 4.22: Lokalisieren und Ausführen von Kommandos

#### 7.1 Shellprogramm führt den entsprechenden Programmteil aus:

Das Shellprogramm **sh** verzweigt zu dem entsprechenden Programmteil, der dieses Kommando realisiert, und führt ihn aus. Wichtig ist, daß hierbei keine neue Subshell gestartet wird, da das entsprechende »Kommando« vom aktuellen Shellprozeß selbst ausgeführt wird.

Anmerkung: Der Ablaufplan aus Abbildung 4.22 verdeutlicht nochmals, warum builtin-Kommandos niemals durch Funktionsdefinitionen ersetzt werden können.

#### 7.2 Shell führt die entsprechende Funktion aus:

Die durch die entsprechende Funktion definierten Kommandos werden ausgeführt, wobei innerhalb der Funktion auf eventuell angegebene Funktionsargumente über die Positionsparameter **\$1**, **\$2**, ... zugegriffen werden kann.

#### 7.3 Shell startet einen neuen Prozeß:

Dazu muß zuerst die entsprechende Programmdatei lokalisiert werden, welche das betreffende Kommando enthält. Für diesen Suchprozeß werden 3 Möglichkeiten unterschieden:

- Kommando ist als absoluter Pfadname (erstes Zeichen ist ein Slash /) angegeben:

Dieser absolute Pfadname zeigt bereits auf die entsprechende Programmdatei, die zu starten ist; z.B. würde der Aufruf



/bin/who

das Programm **who** im Directory **/bin** starten.

- b) Kommando ist als relativer Pfadname angegeben, in dem mindestens ein Slash / vorkommt:

Ausgehend vom Working-Directory wird über »Auf- und Absteigen im Directorybaum« die entsprechende Datei lokalisiert, die zu starten ist; z.B. würde der Aufruf

../bin/psuser

das Programm **psuser** im Subdirectory **bin** des parent directorys starten.

- c) Kommando ist als einfacher Dateiname ohne einen Slash / angegeben:

Es wird in den Directories, welche in der Shellvariablen **PATH** vorgegeben sind, nach dem angegebenen Dateinamen gesucht<sup>1</sup>.

Sobald in einer der durch **PATH** vorgegebenen Directories ein passender Dateiname gefunden werden konnte, wird die Suche abgebrochen.

Ist die gefundene Datei ausführbar und enthält ausführbaren Maschinencode, dann wird diese Programmdatei unverzüglich gestartet.

Ist diese Datei zwar ausführbar, aber enthält keinen ausführbaren Maschinencode, so wird angenommen, daß es sich hierbei um ein Shell-Skript handelt; in diesem Fall wird eine Subshell gestartet, um die in dieser Datei angegebenen Kommandos zu interpretieren und auszuführen.

### Anmerkung zum Lokalisieren einer Datei

Die Shell verwendet ein sogenanntes »hashing«-Verfahren, um sich an einmal gefundene Kommandos auch später wieder erinnern zu können, ohne wieder von neuem die **PATH**-Directories durchsuchen zu müssen; dazu unterhält sie eine Tabelle, in welche die absoluten Pfadnamen aller einmal gefundenen Kommandos festgehalten werden. Dieses »hashing«-Verfahren bewirkt nun, daß die Shell die Variable **PATH** vollständig ignoriert, wenn das angegebene Kommando in dieser Tabelle bereits eingetragen ist<sup>2</sup>.

---

1. Eine genaue Beschreibung dazu wurde in Kapitel 4.6.2 bei der Vorstellung der Shell-Variablen **PATH** gegeben.  
2. Siehe Kapitel 4.15.5, wo das hashing-Verfahren der Shell vorgestellt wurde.

```

Beispiel $ pwd
/user1/egon/shellueb
$ STD="/usr/include/stdio.h"
$ cat such
printf
$ ls p*
pd
psuser
$ ls *.c
addiere.c
div_null.c
hallo.c
$ grep `cat such` *.c $S\
> TD >p*
$ ls p*
p*
pd
psuser
$ cat p*
addiere.c: printf("%d + %d = %d\n", a, b, c);
div_null.c: printf("Die Division von %d / %d = %d\n", a, b, divi);
hallo.c: printf("Hallo egon\n");
/usr/include/stdio.h: getw(), pclose(), printf(), fprintf(), sprintf(),
/usr/include/stdio.h: vprintf(), vfprintf(), vsprintf(), fputc(), putw(),
$

```

### Die Abarbeitung der Kommandozeile

```

$ grep `cat such` *.c $S\
> TD >p*
$

```

wurde von der Shell in folgenden Schritten durchgeführt:

1. Entfernen aller \Neuzeile-Zeichen:

```
grep `cat such` *.c $STD >p*
```

2. Parametersubstitution:

```
grep `cat such` *.c /usr/include/stdio.h >p*
```

3. Kommandosubstitution

```
grep printf *.c /usr/include/stdio.h >p*
```

4. Zerlegen der Kommandozeile in einzelne Worte:

|                            |                             |
|----------------------------|-----------------------------|
| 0. Wort (Kommando):        | <b>grep</b>                 |
| 1. Wort:                   | <b>printf</b>               |
| 2. Wort:                   | <b>*.c</b>                  |
| 3. Wort:                   | <b>/usr/include/stdio.h</b> |
| 4. Wort (Standardausgabe): | <b>p*</b>                   |

### 5. Auswerten der Ein-/Ausgabe-Umlenkungsangaben

Bei Ausgabeumlenkung findet keine Dateinamenexpansion statt. Deshalb wird hier die Standardausgabe für das Kommando **grep** von der Shell in die Datei *p\** umgelenkt

### 6. Expansion von Dateinamen

```
grep printf addiere.c div_null.c hallo.c /usr/include/stdio.h
```

Im zweiten Wort (*\*.c*) wurde das Expandierungs-Metazeichen *\** gefunden, was zu obiger Expansion führt.

### 7. Shell lokalisiert entsprechendes Kommando und führt es aus:

Da es sich bei dem Kommando **grep** weder um ein builtin-Kommando noch um eine Funktion handelt, wird zunächst geprüft, ob **grep** bedingt durch einen vorherigen Aufruf bereits in der hashing-Tabelle eingetragen wurde. Wenn nicht, dann wird es über die **PATH**-Directories gesucht; in diesem Fall wird die zugehörige Programmdatei in **/bin/grep** gefunden. Hierfür wird dann von der Shell ein Kindprozeß gestartet, der mit einem **exec**-Systemaufruf<sup>1</sup> überlagert wird.

Diese Abarbeitungsreihenfolge muß der Benutzer bei einem Kommandoaufruf berücksichtigen; zudem sollte er beachten, daß alle Auswertungen nur einmal durchgeführt werden.

#### Beispiel

```
$ pwd ↵
/user1/egon/shellueb
$ dir=/user1/egon/shellueb/verz ↵
$ cat $dir ↵
/user1/egon/bin
$ cd `cat $dir` ↵
$ pwd ↵
/user1/egon/bin
$ cd ../shellueb ↵
$
```

## 4.17 Die Aufrufsyntax der Shell

Die mögliche Aufrufsyntax für die Bourne-Shell ist:

```
sh [-aefhiknrtuvx] [argument(e)]
```

```
sh [-aefhiknrtuvx] [-c argument]
```

```
sh [-aefhiknrtuvx] [-s] [argument(e)]
```

---

1. Siehe erstes Buch "Linux/Unix-Grundlagen".

Die Login-Prozedur ruft die Shell mit

```
exec -sh
```

auf. Das Minuszeichen - bewirkt, daß in diesem Fall zuerst die Kommandos aus der Datei `/etc/profile` und dann die aus `$HOME/.profile` – falls diese Dateien existieren – ausgeführt werden.

Wenn **sh** mit der Angabe von *argument(e)*, aber ohne die Optionen **-c** und **-s**, aufgerufen wird, dann wird das erste Argument als eine Programmdatei interpretiert, welche entweder ein UNIX-Kommando, ein ausführbares Benutzerprogramm oder ein Shell-Skript sein kann; die restlichen *argument(e)* werden diesem Programm dann als eigentliche Argumente übergeben.

Wird **sh** ohne *argument(e)* aufgerufen, dann wird eine Subshell gestartet, welche Kommandos interaktiv von der Standardeingabe liest. Eine solche Subshell kann immer mit dem Kommando **exit** oder der Eingabe von **EOF** (*Strg-D*) verlassen werden, was dann eine Rückkehr in die Elternshell bewirkt.

Alle beim builtin-Kommando **set** vorgestellten Optionen können auch beim Aufruf von **sh** angegeben werden; diese Optionen gelten dann für die Subshell, die durch **sh** kreiert wird. Die bei **set** erlaubten Optionen werden hier nochmals zum Zwecke der Wiederholung gegeben:

| Optionen  | Bedeutung                                                                                                                                                              |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-a</b> | Shell-Variablen, die verändert oder neu angelegt werden, für den Export markieren.                                                                                     |
| <b>-e</b> | ( <i>exit</i> ) Shell bei nicht erfolgreicher Ausführung eines Kommandos sofort beenden.                                                                               |
| <b>-f</b> | ( <i>filename expansion</i> ) Dateinamenexpansion ausschalten.                                                                                                         |
| <b>-h</b> | ( <i>hash</i> ) Kommandos in einer Funktion schon bei der Definition und nicht erst bei der Ausführung der Funktion lokalisieren und in der Hashing-Tabelle eintragen. |
| <b>-k</b> | ( <i>keywords</i> ) <b>Alle</b> Shell-Variablen an die Umgebung ( <i>environment</i> ) eines Kommandos übergeben.                                                      |
| <b>-n</b> | ( <i>no execution</i> ) Kommandos nur lesen und auf Syntaxfehler untersuchen, aber nicht ausführen.                                                                    |
| <b>-t</b> | ( <i>terminate</i> ) Nach dem Lesen und der Ausführung eines Kommandos entsprechende Shell beenden.                                                                    |
| <b>-u</b> | ( <i>unset</i> ) Zugriff auf nicht gesetzte Shell-Variable als Fehler werten.                                                                                          |
| <b>-v</b> | ( <i>verbose</i> ) Alle Shell-Eingabezeilen – so wie sie gelesen werden – ausgeben.                                                                                    |
| <b>-x</b> | ( <i>execute</i> ) Alle Kommandos und ihre Argumente – so wie sie wirklich von der Shell ausgeführt werden – ausgeben.                                                 |

Zusätzlich können noch die folgenden Optionen angegeben werden:

| Optionen                         | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-c</b> <i>argument</i>        | Die als <i>argument</i> angegebene Kommandoliste wird gelesen und ausgeführt.<br><b>Beispiel:</b><br><code>sh -c 'ls   wc -l'</code><br>gibt die Anzahl von Dateien im Working-Directory aus.                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>-s</b> [ <i>argument(e)</i> ] | bewirkt, daß eine interaktive Subshell gestartet wird, die Kommandos von der Standardeingabe liest. Wenn <i>argument(e)</i> angegeben sind, dann werden diese der Subshell als Positionsparameter übergeben. Ausgaben der Shell werden in diesem Fall auf die mit Dateideskriptor 2 spezifizierte Datei (Standardfehlerausgabe) geschrieben.<br><b>Beispiel:</b><br><code>\$ pwd</code><br>/user1/egon/shellueb<br><code>\$ sh -s egon `pwd`</code><br><code>\$ echo "Hallo, \$1"</code><br>Hallo, egon<br><code>\$ echo \$2</code><br>/user1/egon/shellueb<br><code>\$ exit</code><br>\$ |
| <b>-i</b>                        | macht die Shell interaktiv, d.h., daß die Standardeingabe und die beiden Ausgabekanäle auf die Dialogstation eingestellt sind. Zudem gilt:<br><i>terminate</i> -Signal (Signalnummer 0) wird in dieser Shell ignoriert <sup>a</sup> .<br><i>intr</i> -Signal (Signalnummer 2) wird ignoriert.<br><i>quit</i> -Signal (Signalnummer 3) wird ignoriert.                                                                                                                                                                                                                                     |
| <b>-r</b>                        | die entsprechende Shell wird als »restricted Shell« (eingeschränkte Shell) <sup>b</sup> aufgerufen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

a. Eine solche interaktive Shell kann deshalb nicht mit **kill 0 shell-pid** abgebrochen werden.

b. Siehe Kapitel 4.20.

## 4.18 Handhabung von überlangen Kommandozeilen

Es wurde bereits früher erwähnt<sup>1</sup>, daß manche Kommandozeilen für die Verarbeitung durch die Shell zu lang werden können. Typische Konstruktionen, bei denen solche überlangen Kommandozeilen entstehen können, sind z.B.

```
for i in * oder
grep begriff `find directory -print`
```

Die Kommandosubstitution und Dateinamenexpansion der Shell kann bei solchen Angaben dazu führen, daß die neu entstandene Argumenten-Liste für die Shell oder das aufgerufene Kommando zu lang wird. Dies wird durch die Ausgabe einer Fehlermeldung, wie z.B.

```
sh: arg list too long
```

oder

```
kommandoname: arg list too long
```

angezeigt.

Glücklicherweise treten solche überlangen Kommandozeilen nur selten auf. Wenn sie aber vorkommen, so sollte der Benutzer die Fähigkeit besitzen, dieses Problem zu beseitigen.

Nachfolgend werden anhand eines Beispiels Verfahren gezeigt, die es ermöglichen, das Entstehen von überlangen Kommandozeilen zu vermeiden:

```
$ grep -sn getchar `find /usr/include -print`
/bin/grep: arg list too long
$
```

### Kommando-Ausgabe über Pipe an while-Schleife weiterleiten

```
$ (find /usr/include -print | while read datei
> do
> grep -sn getchar $datei ""
> done; times)
/usr/include/stdio.h:77:#define getchar()getc(stdin)
0m10s 0m33s
$
```

Diese Vorgehensweise hat neben der mehrzeiligen Eingabe den weiteren Nachteil, daß für jede einzelne Datei immer wieder das Kommando **grep** neu aufgerufen wird; dies macht sich natürlich an der gebrauchten CPU-Zeit bemerkbar.

---

1. Bei der Vorstellung der **while**-Schleife in Kapitel 4.12.3.

### Verwendung von **find** mit der **exec**-Bedingung

```
$ (find /usr/include -exec grep -sn getchar {} "" \; ; times)
/usr/include/stdio.h:77:#define getchar()getc(stdin)
0m7s 0m28s
$
```

Bei diesem Verfahren reicht zwar ein einzeliger Kommandoaufruf aus. Trotzdem wird auch hier wieder für jede einzelne Datei das Kommando **grep** neu aufgerufen, was wieder an der verbrauchten CPU-Zeit erkennbar ist. Die etwas verbesserte Ablaufzeit gegenüber dem ersten Verfahren läßt sich hier durch den Wegfall der Pipe erklären.

### Verwendung von **xargs**

```
$ (find /usr/include -print | xargs grep -sn getchar; times)
/usr/include/stdio.h:77:#define getchar()getc(stdin)
0m7s 0m11s
$
```

Die Verwendung von **xargs** bietet eindeutige Vorteile bezüglich der Ablaufzeit eines solchen Kommandos. **xargs** baut dynamisch Kommandozeilen auf, indem es die Argumente für das angegebene Kommando (hier **grep**) von der Standardeingabe liest. Um ein »Überlaufen« einer Kommandozeile zu verhindern, benutzt **xargs** einen internen Puffer, in dem es alle von der Standardeingabe gelesenen Argumente speichert. Wenn dieser Puffer voll ist, ruft es das angegebene Kommando (hier **grep**) mit diesen Argumenten auf. Danach wird dieser ganze Prozeß wiederholt, bis keine Argumente mehr in der Standardeingabe vorhanden sind.

Eine ausführlichere Beschreibung zu **xargs** wird im Buch »Linux-Unix-Kurzreferenz« gegeben. Hier soll nun ein Beispiel zu **xargs** gegeben werden:

Es ist ein ganzer Directorybaum umzukopieren (zu verlagern):

Eine mögliche Lösung zu dieser Aufgabenstellung wäre z.B. der Aufruf

```
find quelldir -print | cpio -pd zieldir
```

Ein Nachteil hierbei ist, daß bei eventuellem Speicherplatzmangel dieses Kommando abbricht und nur ein Teil des Quell-Directorybaums kopiert würde. Zudem muß anschließend immer mit

```
rm -rf quelldir
```

der Quell-Directorybaum entfernt werden.

Eine andere Möglichkeit wäre die Verwendung des Kommandos **xargs**:

```
$ pwd
/user1/egon/shellueb
$ cat mvbaum
mvbaum Version v1.0 (20.1.1997)
verlagert den Directorybaum quelldir nach zieldir
#
Syntax: mvbaum quelldir zieldir
#
Autor: Egon ...
#
if [$# -ne 2]
then
 echo "$0: falsche Anzahl von Argumenten" >&2
 echo "usage: mvbaum quelldir zieldir" >&2
 exit 1
fi

if [! -d $1]
then
 echo "$0: $1 ist kein directory" >&2
 echo "usage: mvbaum quelldir zieldir" >&2
 exit 1
fi

QUELLE=$1
ZIEL=$2

if [-d $ZIEL]
then
 echo "\"$ZIEL\" existiert bereits. Loeschen ? (J/N) : \c"
 read ANTWORT
 if ["$ANTWORT" = "j" -o "$ANTWORT" = "J"]
 then
 rm -rf $ZIEL
 mkdir $ZIEL
 fi
else
 mkdir $ZIEL
fi

cd $QUELLE # Wechseln ins Quell-Directory

(1) Directory-Baumsstruktur von $QUELLE in $ZIEL herstellen
-i bewirkt, dass Kommando mkdir fuer jedes von find gelieferte
Argument aufgerufen wird.
#
(2) Alle einfachen Dateien aus $QUELLE in die durch (1) kreierte
Directory-Baumstruktur von $ZIEL verlagern.
#
(3) Entfernen des Quell-Directorybaums
```



```
find * -type d -print | xargs -i mkdir $ZIEL/{} && # (1)
find . -type f -print | xargs -i mv {} $ZIEL/{} && # (2)
rm -rf * # (3)
$ chmod u+x mvbaum
$
```

Die Angabe der Option **-i** beim Aufruf von **xargs** bewirkt, daß das entsprechende Kommando für jedes einzelne aus der Pipe gelesene Argument aufgerufen wird. Die Angabe des Klammersnpaares {} bewirkt, daß dieses Argument beim Aufbau der Kommandozeile eingesetzt wird. Nun soll das Shell-Skript *mvbaum* getestet werden, indem zunächst mit dem früher geschriebenen Shell-Skript *cpdir* der Directorybaum **\$HOME/bin** nach **\$HOME/bin2** kopiert wird. Danach wird dann mit *mvbaum* der Directorybaum **\$HOME/bin2** nach **\$HOME/bin3** verlagert:

```
$ pwd
/user1/egon/shellueb
$ cd ..
$ cpdir bin bin2
36 blocks
$ cd shellueb
$ mvbaum ../bin2 ../bin3
$ lc ../bin
./ cpdir* erinner* holedat* suchtext* zdrucke*
../ cpzdr* gebdat* lc* wartuser*
bildsperr* datlaufe* gruesse* letztlog* wer*
copy* del* hexa* psuser* werist*
$ lc ../bin3
./ cpdir* erinner* holedat* suchtext* zdrucke*
../ cpzdr* gebdat* lc* wartuser*
bildsperr* datlaufe* gruesse* letztlog* wer*
copy* del* hexa* psuser* werist*
$ lc ../bin2
./ ./
$ cp mvbaum ../bin
$
```

Das Kommando **xargs** ist also ein nützliches Werkzeug, um Aufrufe, die in ihrer herkömmlichen Form in überlangen Kommandozeilen resultieren, so umzuformen, daß sie von der Shell oder vom jeweiligen Kommando verarbeitet werden können. Zudem kann die Verwendung von **xargs** – wie zuvor gezeigt wurde – zu erheblichen Laufzeit-Verbesserungen führen, da **xargs** die mögliche Länge einer Kommandozeile optimal ausnützt und nicht unbedingt für jedes Argument einen neuen Prozeß startet. **xargs** ist sicher kein häufig verwendetes Kommando wie **grep** oder **find**. Trotzdem ist es gut zu wissen, daß es dieses Kommando gibt, um es im Bedarfsfall benutzen zu können.

## 4.19 Die Datei *.profile*

Wenn ein Benutzer sich am System anmeldet, so startet die login-Prozedur unter Verwendung einer **exec**-Funktion<sup>1</sup> die Shell mit *-sh*; das führende Minuszeichen zeigt an, daß es sich um eine Login-Shell handelt. Deshalb führt sie zuerst die Datei */etc/profile*<sup>2</sup> aus und dann die Datei *.profile* im Home-Directory des speziellen Benutzers. Jede neue Shell, die nach dieser Login-Shell gestartet wird, führt keine dieser *profile*-Dateien mehr aus. Der Inhalt der Datei */etc/profile* kann Informationen über Maschineneigenschaften und Voreinstellungen für alle Benutzer geben. Ein Beispiel für das Aussehen einer solchen Datei */etc/profile* könnte sein:

```
$ cat /etc/profile ↵
#
.....
Kommentar
.....

trap "" 1 2 3
umask 022 # set default file creation mask
. /etc/TIMEZONE

case "$0" in
-sh | -rsh)
 trap : 1 2 3
 if [-s /etc/motd] ; then cat /etc/motd; fi # issue message of the day
 trap "" 1 2 3
 stty erase '^h' echoe
 case `tty` in
 /dev/tty*) TERM=vt100 ; break ;;
 *) TERM=AT386 ; break ;;
 esac
 export TERM;

 # check mailbox and news bulletins
 if mail -e
 then echo "you have mail"
 fi
 if [$LOGNAME != root]
 then news -n
 fi
 ;;
-su)
 :
 ;;
esac
export PATH;
trap 1 2 3
$
```

---

1. Siehe »Linux-Unix-Grundlagen«.

2. Die vom Systemverwalter eingerichtete allgemeine *profile*-Datei.

In der Datei *.profile* (im Home-Directory) werden üblicherweise Variablen oder Funktionen definiert, die zum einen eine spezielle Shell-Umgebung für den Benutzer festlegen und zum anderen auch das spätere Arbeiten mit der Shell erleichtern. Damit solche Shell-Variablen auch in späteren Subshells bekannt sind, sollten diese mit dem Kommando **export** exportiert werden. Um ein Aufblähen der Datei *.profile* zu vermeiden ist es üblich, die Funktionsdefinitionen in einer anderen Datei im Home-Directory (hier *funkdef*) unterzubringen und diese dann mit dem Punkt-Kommando in *.profile* zu lesen. Weitere typische Aktionen in *.profile* sind z.B.:

- ▶ Festlegen einer benutzereigenen Signalbehandlung für die Login-Shell.
- ▶ Festlegen von Aktionen, die immer am Anfang einer UNIX-Sitzung durchzuführen sind, wie z.B. Ausgabe des letzten Anmeldezeitpunkts.
- ▶ Abgeben von **cron**- und **at**-Aufträgen, wie z.B. Einbau eines automatischen Erinnerungs-Service.

Wird der Inhalt von *.profile* während einer UNIX-Sitzung geändert, ist damit nicht auch automatisch die entsprechende Shell-Umgebung geändert. Dazu muß die Shell zunächst den Inhalt von *.profile* wieder neu lesen. Am einfachsten erreicht man dies unter Verwendung des Punkt-Kommandos:

```
. .profile
```

Nachfolgend wird ein Beispiel für das Aussehen einer *.profile*-Datei und einer begleitenden Funktionsdefinitions-Datei (hier *funkdef*) gegeben. Dies ist lediglich als Vorschlag zu verstehen. Jeder Benutzer kann dann diese beiden Dateien weiter ausbauen und ändern, um sie seinen speziellen Bedürfnissen anzupassen:

```
$ cat $HOME/.profile 
#
.....
Kommentar
.....

Definition von nuetzlichen Variablen
hu=$HOME/shellueb
hb=$HOME/bin
dirs="hu hb"
export $dirs dirs

Lesen von nuetzlichen Funktionsdefinitionen
. $HOME/funkdef

Definition von vordefinierten Shell-Variablen
MAIL=/usr/mail/${LOGNAME:?}
MAILCHECK=0
PATH=/bin:/usr/bin:$HOME/bin:/etc:.
TERM=vt100
CDPATH=...:$HOME:$dirs
```

4.19 Die Datei *.profile*

225

```
export MAIL MAILCHECK PATH TERM CDPATH
tput init

Ausgabe des letzten Anmeldezeitpunkts
letztlog -m; letztlog

Ausgabe des Abmeldezeitpunkts
trap 'echo "`pwd` `date`" >${HOME}/.zuletzt' 0
if [-f ${HOME}/.zuletzt]
then
 read workdir datum <${HOME}/.zuletzt
 echo "**** Letzte Abmeldung war: $datum ****"
fi

Ausgabe des heutigen Datums und zeitabhaengige Meldung
echo "\n**** Heute ist `date` ****\n"
gruesse

Erinnerungs-Service
/usr/bin/calendar

Wechseln zum letzten work. dir und Ausgabe des Working-Directorys
if [-n "$workdir"]
then
 cd $workdir
fi
echo "\n`pwd`>\c"
$ cat ${HOME}/funkdef 
#-- ll -----
ruft ls -CF auf und gibt Anzahl der aufgelisteten Dateinamen aus
ll() {
 ls -CF "$@"
 echo "-----\n`ls "$@" | wc -l` Dateien"
}

#-- cdir -----
entspricht dem Kommando cd, allerdings wird danach das
Working-Directory als Prompt-String ausgegeben
cdir() {
 cd $1
 PS1="`pwd`> "
}

#-- zaehle -----
realisiert eine for-Schleife mit step-Angabe
1.Argument: Startwert
2.Argument: Endwert
3.Argument: Schrittweite
zaehle() {
```

```
start=${1:-1}
ende=${2:-10}
step=${3:-1}
i=$start
while [$i -le $ende]
do
 echo $i
 i=`expr $i + $step`
done
}

#-- rm -----
loescht Dateien nicht sofort, sondern fragt nach
rm() {
 case "$1" in
 -f|-r|-i) /bin/rm $*;;
) /bin/rm -i $;;
 esac
}

#-- pushd -----
wechselt - wie cdir - zum angegebenen Directory.
das aktuelle Directory wird allerdings in der Variablen
PUSHHD festgehalten, so dass mit popd immer zum vorherigen
Directory zurueckgekehrt werden kann.
Wird kein Directory beim Aufruf angegeben, so gibt pushd
den gemerkten "Directory-Stack" aus.
pushd() {
 if [$# -gt 1]
 then
 echo "usage: pushd [directory]" >&2
 return 1
 elif [$# -eq 0]
 then
 echo "$PUSHHD"
 return 0
 fi

 if [-z "$tiefe"]
 then
 tiefe=0
 fi
 if [-d "$1" -a -x "$1"]
 then
 PUSHHD=`pwd` $PUSHHD
 tiefe=`expr $tiefe + 1`
 cdir "$1"
 else
 echo "Kann nicht zu $1 wechseln" >&2
 fi
}
```

```
 return 2
 fi
}

#-- popd -----
ist das Gegenstueck zu pushd: Die als erstes Argument
angegebene Zahl legt fest, um wie viele Directories auf
dem "Directory-Stack" zurueckgekehrt werden soll.
Wird keine Zahl angegeben, so wird zur obersten Directory
des "Directory-Stacks" zurueckgekehrt.
Wird popd - aufgerufen, so wird der ganze "Directory-Stack"
geleert und wieder der default-Prompt eingestellt
popd() {
 if ["$1" = "-"]
 then
 unset PUSHD tiefe
 PS1="$ "
 return 0
 fi
 zahl=${1:-1}
 if ["$tiefe" -lt "$zahl" -o "$tiefe" -lt 1]
 then
 echo "zu kleiner Directory-Stack" >&2
 return 1
 fi
 d=`echo "$PUSHD" | cut -d" " -f${zahl} -`
 cd "$d"
 s=`expr $zahl + 1`
 PUSHD=`echo "$PUSHD" | cut -d" " -f${s} -`
 tiefe=`expr $tiefe - $zahl`
}
$
```

## 4.20 Die eingeschränkte Shell rsh

Oft befinden sich unter den Benutzern einer großen Rechananlage auch solche, denen nicht der volle Zugang zu allen Daten und Kommandos in einem System erlaubt sein soll (z.B. Werkstudenten, Mitarbeitern aus Fremdfirmen usw.). Für solche Benutzer wurde die eingeschränkte Shell **rsh** (*restricted shell*) erfunden.

Obwohl **rsh** ebenso wie **sh** explizit als eigener Prozeß gestartet werden kann, so wird **rsh** doch meist durch einen entsprechenden Eintrag in */etc/passwd* als Login-Shell für solche »zweitklassige Benutzer« gestartet.

**rsh** hat die gleiche Aufrufsyntax und die gleichen Optionen wie **sh**<sup>1</sup>:

---

1. Siehe Kapitel 4.17.

```
rsh [-aefhiknrtnvx] [argument(e)]
```

```
rsh [-aefhiknrtnvx] [-c argument]
```

```
rsh [-aefhiknrtnvx] [-s] [argument(e)]
```

Nach dem Starten werden die Kommandos aus den Dateien */etc/profile*, */etc/rsh* und *.profile* (im Home-Directory des entsprechenden Benutzers) ausgeführt. Erst danach werden die für **rsh** vorgegebenen Einschränkungen wirksam.

Die eingeschränkte Shell **rsh** verhält sich bis auf die folgenden Einschränkungen genauso wie die Shell **sh**:

- ▶ Kommando **cd** ist nicht erlaubt:  
Der entsprechende Benutzer kann nur in dem ihm zugeteilten Working-Directory arbeiten.
- ▶ Inhalt der vorgegebenen Shellvariablen **PATH** darf nicht verändert werden:  
Damit ist sichergestellt, daß der entsprechende Benutzer nur Programme ausführen darf, die ihm über die vorgegebenen **PATH**-Directories zugestanden werden.
- ▶ Pfad- und Kommandonamen, welche / enthalten, können nicht verwendet werden.
- ▶ Die Ausgabe-Umlenkungen > und >> können nicht verwendet werden.

Die Einschränkungen für Pfadnamen, für das **cd**-Kommando und für die **PATH**-Variable ermöglichen es dem Systemadministrator, dem entsprechenden Benutzer eine Shellumgebung einzurichten, in der er nur Zugriff auf eine festumrissene Menge von Kommandos hat. Beispielsweise könnte der Systemadministrator ein Directory */usr/rbin* anlegen, welches nur Links auf die gestatteten Kommandos enthält. Dem »eingeschränkten Benutzer« würde dann z.B. anstelle von */usr/bin* dieses Directory */usr/rbin* in die **PATH**-Variable eingetragen. Die Datei *.profile* für einen solchen Benutzer würde dann vom Systemadministrator erstellt und könnte z.B. wie folgt aussehen:

```
$ cat .profile
PATH=/usr/rbin
SHELL=/bin/rsh
export PATH SHELL
cd /user1/rsh/rudi
$
```

Mit diesem *.profile* wird erreicht, daß der entsprechende Benutzer nur die Kommandos aus */usr/rbin* aufrufen kann. Versucht er andere Kommandos aufzurufen, so wird ihm z.B. gemeldet:

*kommandoname: not found*

Außerdem wird er in das Directory `/user1/rsh/rudi` verbannt, das er niemals verlassen kann, da ihm **rsh** nicht die Verwendung des **cd**-Kommandos zugesteht. Da dieses Directory nicht die Datei `.profile` enthält, besteht auch keine Möglichkeit für ihn, den Inhalt dieser Datei zu ändern.

**Hinweise**

1. Der Aufruf von **rsh** kann auf vier verschiedenen Arten erfolgen:
  - (1) Als Login-Shell durch entsprechenden Eintrag in `/etc/passwd`; dies ist die meist vorkommende Aufrufform.
  - (2) Beim Aufruf einer Shell existiert die vordefinierte Shell-Variable **SHELL** und enthält als Wert einen Pfadnamen, dessen Basisname `rsh` ist.
  - (3) Durch direkten Aufruf von **rsh**.
  - (4) Das Programm **sh** wird mit der Option **-r** aufgerufen.
2. Es sollte einem »eingeschränkten Benutzer« niemals Zugriff auf ein Shell-Programm<sup>1</sup>, einen Compiler oder das Kommando **chmod** gewährt werden. Ein erfahrener UNIX-Anwender könnte mit Hilfe dieser Kommandos eventuell die vorgegebenen Restriktionen umgehen.
3. Zum Editor **ed** existiert auch eine eingeschränkte Version **red** (*restricted ed*). **red** entspricht dem **ed**, mit dem Unterschied, daß ein Wechseln in eine andere Shell nicht möglich ist und nur Dateien im Working-Directory editiert werden können.
4. Nun kann es vorkommen, daß manche »zweitklassige Benutzer« Kommandos benötigen, die für die Dauer ihrer Ausführung diese Einschränkung umgehen<sup>2</sup>; für diesen Fall ist es üblich, dem Benutzer diese Kommandos in Form von Shell-Skripts zur Verfügung zu stellen. Zur Ausführung eines Shell-Skripts ruft nämlich **rsh** die normale Shell **sh** auf, welche keine Einschränkungen kennt. Um bei einer solchen Vorgehensweise zu verhindern, daß ein Benutzer sich über eigene Shell-Skripts Zugang zur normalen Shell-Welt verschafft, sollte dieser im Working-Directory weder Schreib- noch Ausführrechte besitzen.

## 4.21 Job-Kontrolle mit sh1

Bis zu System V gab es keine Möglichkeit einer Job-Kontrolle, wie sie von Berkeley-UNIX her bekannt ist. Unter Job-Kontrolle versteht man z.B., daß es dem Benutzer erlaubt ist, ein Programm vorübergehend anzuhalten und an späterer Stelle wieder zu starten. Mit System V.2 wurde nun das Kommando **sh1** eingeführt, welches eine einfache Job-Kontrolle ermöglicht. Die Aufrufsyntax ist dabei:

- 
1. `/bin/sh`, `/bin/csh`, `/bin/ksh`, `/bin/bash` oder `/bin/tcsh`.
  2. Beispiele für solche Kommandos sind **env**, **cp** oder **ln**.



`shl` (ohne jegliche Argumente oder Optionen)

Unter den meisten heutigen UNIX-Systemen wird diese doch veraltete Technik nicht mehr benötigt. Unter Linux z.B. kann man sehr leicht mit den Tasten *Alt-F1*, *Alt-F2* usw. zwischen virtuellen Terminals hin- und herschalten.

### Beschreibung

Während UNIX-System V das gleichzeitige Ablaufen mehrerer Hintergrundjobs erlaubt, läßt es dagegen normalerweise nur eine interaktive Shell im Vordergrund zu. Mit dem Kommando **shl** ist es nun möglich, mehrere interaktive Shells (auch *layers* genannt) gleichzeitig ablaufen zu lassen. Dabei kann zu einem Zeitpunkt immer nur ein Layer aktiv (interaktiv) sein; allerdings ist es möglich, zwischen den verschiedenen Layers hin- und herzuschalten.

### Layers und virtuelle Terminals

Eine normale interaktive Shell ist immer einem bestimmten physikalisch vorhandenen Terminal zugeordnet; auf dieses Terminal ist dann die Standardeingabe, Standardausgabe und Standardfehlerausgabe eingestellt.

Ein Layer ist eine Shell, welche einem »virtuellen Terminal« zugeordnet ist; ein virtuelles Terminal ist kein physikalisch existierendes Terminal, sondern ein imaginäres Terminal, das nur in der Vorstellung der Software existiert. Die Einstellungen für ein solches virtuelles Terminal können allerdings genauso wie die für ein wirkliches Terminal unter Verwendung des Kommandos **stty** geändert werden. Jeder Layer besitzt sein eigenes virtuelles Terminal. Für jede existierende Dialogstation sind maximal bis zu 7 Layers (und den zugehörigen virtuellen Terminals) zugelassen. Das Kommando **shl** erlaubt es, einen Layer zum aktiven Layer (engl. *current layer*) zu machen. Damit wird das diesem Layer gehörige virtuelle Terminal dem wirklichen Terminal zugeordnet. Somit ist der gerade aktive Layer immer der Layer, der Eingaben über die Tastatur entgegennimmt. Andere, nicht aktive Layer, die Eingaben erwarten, werden dabei blockiert, bis auf sie umgeschaltet wird, und sie damit zum aktiven Layer werden. Ausgaben von nicht aktiven Layers werden dagegen normalerweise nicht blockiert, sondern ausgegeben. Dies kann zu gemischten Bildschirm-Ausgaben von verschiedenen Layers führen. Um einen bestimmten nicht aktiven Layer daran zu hindern, daß er seine Ausgabe auf den Bildschirm schreibt, kann dessen Ausgabe mit

`stty ioblk` [für diese Eingabe muß der entspr. Layer aktiv sein]

blockiert werden.

Eine andere Möglichkeit, die Ausgabe eines Layers zu blockieren, ist die Verwendung des **shl**-Kommandos

`block layer-name(n)`

Solche Ausgabeblockaden können wieder aufgehoben werden mit:

```
stty -ioblk [für diese Eingabe muß der entspr. Layer aktiv sein]
```

oder

```
unblock layer-name(n) [shl-Kommando]
```

## Layer-Manager

Nach dem Aufruf

```
shl
```

befindet sich der Benutzer im **shl**-Kommandomodus (auch Layer-Manager genannt); dies wird durch den Prompt >>> angezeigt. Nun können **shl**-Kommandos eingegeben werden, um z.B. neue Layers zu kreieren (**create**), Layers zu löschen (**delete**), usw. Befindet sich der Benutzer in einem Layer, so befindet er sich damit in der zu diesem Layer gehörigen interaktiven Shell. Er kann nun unter dieser Shell arbeiten<sup>1</sup>, ohne daß davon die Shells der anderen Layers betroffen sind. Vom momentan aktiven Layer kann er jederzeit mit der Eingabe der **switch**-Taste (*Strg-z*)<sup>2</sup> wieder zum **shl**-Kommandomodus (Layer-Manager) zurückkehren (siehe auch Abbildung 4.23).

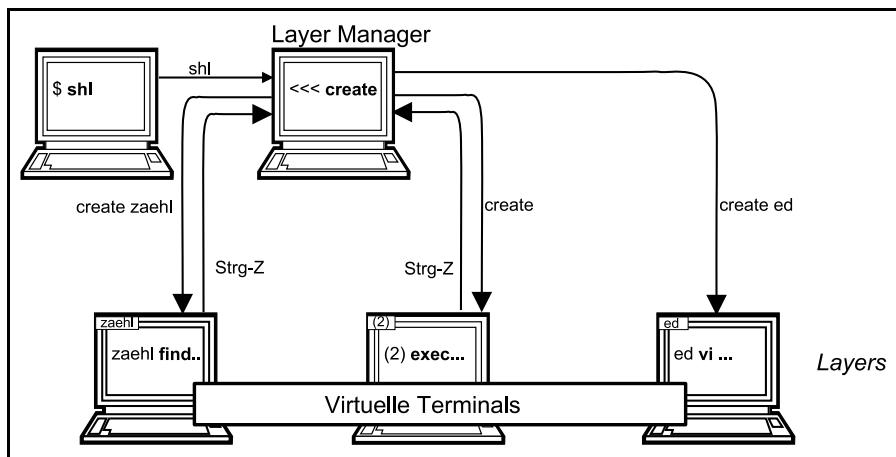


Abbildung 4.23: Layer-Manager und virtuelle Terminals

1. Als Prompt wird dabei immer der Layer-Name ausgegeben.
2. Kann mit dem **stty**-Kommando (siehe »Linux-Unix-Grundlagen«) auch einer anderen Tastenkombination zugeordnet werden.

## shl-Kommandos

Die meisten der **shl**-Kommandos erlauben die Angabe von Layer-Namen. Ein Layer-Name ist dabei eine Zeichenkette, in der kein Leer-, Tabulator- oder Neuezeile-Zeichen vorkommt. Werden bei einem **shl**-Kommando mehrere Layer-Namen angegeben, so sind diese mit Leer- oder Tabulator-Zeichen voneinander zu trennen.

Bei Layer-Namen sind nur die ersten 8 Zeichen signifikant; so würde es sich z.B. bei *terminal1* und *terminal2* um den gleichen Layer-Namen *terminal* handeln.

Die Namen (1), (2), .., (7) sind dabei reserviert. Dies sind die voreingestellten Layer-Namen, wenn beim Anlegen eines neuen Layers nicht explizit eigene Layer-Namen angegeben werden. Obwohl die Klammern zum Namen gehören, so können diese Layers auch mit der Kurzform 1, 2, .., 7 angesprochen werden.

Die folgenden Kommandos können im **shl**-Kommandomodus (Prompt >>>) aufgerufen werden:

| Kommando                        | Beschreibung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>create</b> [name]            | <p>kreiert einen Layer mit den Namen <i>name</i> und macht ihn zum aktiven Layer. Ist kein <i>name</i> angegeben, so wird für diesen neuen Layer einer der Namen (1), (2), .., (7) gewählt; die Namensziffer wird dabei über die letzte Ziffer des virtuellen Terminal-Namens für diesen Layer bestimmt. Die Namen der virtuellen Terminals sind dabei /dev/sxt001, /dev/sxt002, ../dev/sxt007. Daraus ist zu ersehen, daß maximal 7 Layers gleichzeitig vorhanden sein können.</p> <p>Der mit <b>create</b> erzeugte Layer meldet sich mit seinem Namen als Prompt (<b>PS1</b>=<i>layername</i>).</p> |
| <b>block</b> name<br>[name(n)]  | <p>blockiert die Ausgaben der Layers mit den angegebenen <i>name(n)</i>. Das bedeutet, daß diese Layers ihre Ausgaben erst dann fortsetzen können, wenn sie wieder aktiviert werden.</p> <p>Die Ausgaben eines Layers können auch blockiert werden, indem dieser zum aktiven Layer gemacht wird und dann mit <b>stty ioblk</b> seine Ausgaben gesperrt werden.</p>                                                                                                                                                                                                                                     |
| <b>delete</b> name<br>[name(n)] | löscht die Layers mit den angegebenen <i>name(n)</i> , indem allen ihren Prozessen das Signal <b>SIGHUP</b> ( <i>hangup</i> ) geschickt wird.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>help</b> oder ?              | gibt alle <b>shl</b> -Kommandos mit ihrer Aufrufsyntax aus.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>layers</b> [-l] [name(n)]    | <p>gibt zu jedem der Layers mit den angegebenen <i>name(n)</i> die Layer-Namen und die Prozeß-Gruppennummer aus.</p> <p>Die Option <b>-l</b> erzeugt eine <b>ps</b>-ähnliche Ausgabe.</p> <p>Sind keine <i>name(n)</i> angegeben, so wird für alle Layers eine entsprechende Information ausgegeben.</p>                                                                                                                                                                                                                                                                                               |

| Kommando                                       | Beschreibung                                                                                                                                                                                                                                                                             |
|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>resume</b> <i>[name]</i>                    | macht den Layer <i>name</i> zum aktiven Layer.<br>Ist kein <i>name</i> angegeben, so wird der Layer aktiviert, der zuletzt aktiver Layer war.                                                                                                                                            |
| <b>toggle</b>                                  | macht den Layer, der vor dem letzten aktiven Layer aktiv war zum aktiven Layer. Wenn z.B. der letzte aktive Layer (4) und der aktive Layer davor (2) war, so macht <b>toggle</b> den Layer (2) zum aktiven Layer.                                                                        |
| <b>unblock</b> <i>name</i><br><i>[name(n)]</i> | hebt die Ausgabe-Blockierung für die über die <i>name(n)</i> spezifizierten Layers auf. Das Aufheben einer Blockierung für einen Layer könnte auch erreicht werden, indem der entsprechende Layer aktiviert wird und seine Ausgabeblockade mit<br><b>stty -ioblk</b><br>aufgehoben wird. |
| <b>quit</b>                                    | bewirkt das Verlassen des Kommandos <b>shl</b> , indem allen Layern das <b>SIGHUP</b> -Signal ( <i>hangup</i> ) geschickt wird.                                                                                                                                                          |
| <i>name</i>                                    | macht den Layer <i>name</i> zum aktiven Layer.                                                                                                                                                                                                                                           |

#### Beispiel

```

$ pwd
/user1/egon/shellueb
$ shl
>>> create zaehl [Kreieren eines Layers zaehl]
zaehl [Strg-Z] [Zurückschalten zum Layer-Manager]
>>> block zaehl [Ausgabeblockade für Layer zaehl einrichten]
>>> resume [letzten aktiven Layer (zaehl) zum aktiven Layer machen]
resuming zaehl
zaehl find / -name "*.c" -print | wc -l [Alle C-Dateien des Dateisystems
 zaehlen]
[Strg-Z] [Zurückschalten zum Layer-Manager]
>>> create [Kreieren eines neuen Layers;
 Name des neuen Layers wird (2)]
(2) exec csh [Überlagere Shell des Layers (2) mit der C-Shell]
% ls z*
zaehle.txt
zdrucke
zdruckel
zeichtyp
zeig_pid
zusamm
% [Strg-Z] [Zurückschalten zum Layer-Manager]
>>> create ed [Kreieren eines Layers ed]
ed PS1="ed$ " [neues Promptzeichen für diesen Layer festlegen]
ed$ vi cx [Editieren der Datei cx]
... [Eingabe des Inhalts von cx (siehe unten)]

```

```
 und vi wieder verlassen]
ed$ cat cx[↵]
chmod u+x $*
ed$ chmod u+x cx[↵]
ed$ [Strg-Z] [Zurückschalten zum Layer-Manager]
>>> layers[↵] [Information zu allen Layers ausgeben lassen]
zaehl (1445) blocked on input
(2) (1457) executing or awaiting input
ed (1563) executing or awaiting input
>>> zaehl[↵] [Layer zaehl zum aktiven Layer machen]
resuming zaehl
 18421 [soviele C-Programmdateien sind im Dateisystem vorhanden]
zaehl [Strg-Z] [Zurückschalten zum Layer-Manager]
>>> delete ed[↵] [Löschen des Layers ed]
>>> layers[↵] [Information zu allen Layers ausgeben lassen]
zaehl (1445) executing or awaiting input
(2) (1457) executing or awaiting input
>>> quit[↵] [shl verlassen]
$
```

## 4.22 Anwendungsbeispiele

Hier werden anhand nützlicher Shell-Skripts Beispiele für die vielseitigen Einsatzmöglichkeiten der Shell gegeben. Die hier vorgestellten Shell-Skripts dienen dabei folgenden Zwecken:

- ▶ Wiederholung, indem sie von möglichst vielen Shell-Konstruktionen Gebrauch machen.
- ▶ Aufzeigen des Zusammenspiels der einzelnen Shell-Konstrukte, indem anhand praxisnaher Shell-Programmierung die Einsatzmöglichkeiten der Shell gezeigt werden; dabei soll verdeutlicht werden, um welch mächtiges Werkzeug es sich bei der Shell handelt, wenn es verstanden wird, ihre vielseitigen Fähigkeiten richtig einzusetzen.
- ▶ Verständnis und Nachvollziehen bereits existierender Skripts. Sehr oft wird ein Benutzer bereits existierende Shell-Skripts vorfinden, die es zu verbessern gilt. In solchen Fällen muß er diese Skripts zunächst verstehen. Da jeder Skript-Schreiber mit der Zeit seinen eigenen Stil entwickelt, der leider nicht immer den Grundsätzen der strukturierten Programmierung, wie etwa gute Lesbarkeit genügt, ist die Fähigkeit des Verstehens bereits existierender Shell-Skripts von größter Bedeutung.

### 4.22.1 Konvertieren von Zahlen (Oktal, Dezimal, Hexadezimal)

Das nachfolgende Shell-Skript *konvert* erlaubt die Konvertierung von Zahlen zwischen dem Oktal-, Zehner- und Hexadezimal-System. Dabei benutzt es das Kommando **bc**. Dieses Kommando simuliert einen Taschenrechner mit beliebiger Genauigkeit. Nach dem Aufruf liest **bc** die zu berechnenden arithmetischen Ausdrücke von der Standardeingabe und gibt dann das Ergebnis auf die Standardausgabe aus:

```
$ bc
4.25 * 4.25 * 3.1415
56.7354
(12+5.3) * 17.6
304.4
3.7^4 [bedeutet: 3.7 hoch 4]
187.4
[Strg-D] [bc wird mit EOF (Strg-D) beendet]
$
```

Neben diesen Grundrechenarten verfügt **bc** noch über andere Fähigkeiten; so läßt sich dieser Rechner beispielsweise in einer C-ähnlichen Sprache programmieren oder führt auch Konvertierungen zwischen verschiedenen Zahlensystemen durch. Zur Konvertierung bietet **bc** zwei builtin-Variablen an: **ibase** (legt die Basis für die eingegebenen Zahlen fest) und **obase** (legt die Basis für auszugebende Zahlen fest). Voreinstellung für beide Variablen ist dabei das Zehnersystem.

```
$ bc
120
120
ibase=2
120
1111000
ibase=16
45054
AFFE
ibase=2
111011110001010
EF8A
ibase=10 [Festlegen von ibase auf 10-er System dual,
da momentan ibase=2 gilt]
123456789012134567890
6B14E9F8122A11FD2
[Strg-D]
$
```

Eine vollständige Beschreibung zu den vielseitigen Fähigkeiten des Kommandos **bc** befindet sich in Kapitel 9.

```
$ pwd
/user1/egon/shellueb
$ cat konvert
konvert Version v1.0 (7.2.1997)
fuehrt Zahlen-Konvertierungen durch
#
Syntax: konvert
#
Autor: Egon ...
#

tput clear # Bildschirm loeschen
while true
do
 cat <<EOF >&2

 Konvertierungs-Moeglichkeiten:

 1 : Dezimal ---> Hexa
 2 : Hexa ---> Dezimal
 3 : Decimal ---> Oktal
 4 : Oktal ---> Dezimal
 5 : Oktal ---> Hexa
 6 : Hexa ---> Oktal
EOF
 echo " Eingabe : \c" >&2
 read eing
 echo "\n\n"

 case $eing in
 "") exit;;
 1) echo "Dezimalzahl (CR fuer Ende) : \c" >&2
 ebasis=10
 abasis=16;;
 2) echo "Hexazahl (keine Kleinbuchstaben; CR fuer Ende) : \c" >&2
 ebasis=16
 abasis=10;;
 3) echo "Dezimalzahl (CR fuer Ende) : \c" >&2
 ebasis=10
 abasis=8;;
 4) echo "Oktalzahl (CR fuer Ende) : \c" >&2
 ebasis=8
 abasis=10;;
 5) echo "Oktalzahl (CR fuer Ende) : \c" >&2
 ebasis=8
 abasis=16;;
 6) echo "Hexazahl (keine Kleinbuchstaben; CR fuer Ende) : \c" >&2
 ebasis=16
 abasis=8;;
 *) echo "\007 $eing ist nicht erlaubt\n\n" >&2
 continue;;
 esac
done
```

```
esac
read zahl
if ["$zahl" = ""]
then
 exit
fi
erg=`bc <<EOF
 obase=$abasis
 ibase=$ebasis
 $zahl
EOF`
echo "\n$zahl ($ebasis) = $erg ($abasis)\n"
done

$ cx konvert ↵
$ cp konvert ../bin ↵
$
```

## 4.22.2 Durchsuchen ganzer Directorybäume mit grep

Das Kommando **grep** führt keine rekursive Suche in allen Subdirectories eines beliebigen Directories durch. Da oft unbekannt ist, in welchen Directories sich die zu durchsuchenden Dateien befinden, müssen diese meist erst mühsam lokalisiert werden, bevor dann das **grep**-Kommando auf sie angewendet werden kann. Deswegen wäre es zuweilen nützlich, wenn man über ein **grep**-ähnliches Kommando verfügt, das automatisch in allen Subdirectories eines Directories die angegebenen Dateien nach einem bestimmten Begriff durchsucht. Das Shell-Skript *baumgrep* erfüllt diese Aufgabe.

Die Aufrufsyntax dieses Shell-Skripts ist dabei ähnlich zu der von **grep**, nur können zusätzlich noch Directories angegeben werden, die zu durchsuchen sind.

Beim Aufruf von *baumgrep* sollten die ersten beiden Argumente, wenn sie Metazeichen beinhalten, mit **'.'** geklammert sein, um eine Interpretation dieser Metazeichen durch die Shell zu unterbinden.

```
$ cat baumgrep ↵
baumgrep Version v1.0 (7.2.1997)
grep-Kommando fuer ganze Directorybaume
#
Syntax: baumgrep '[Optionen] reg.ausdr' 'datei(en)' [directory(s)]
optionen alle fuer grep zugelassenen Optionen
reg.ausdr legt die zu suchenden Begriffe fest
datei(en) legt die zu durchsuchenden Dateien fest;
Expandierungszeichen sind dabei erlaubt.
directorys geben die Wurzeln der zu durchsuchenden
Directorybaeume an; ist kein directory angegeben,
so wird Working-Directory angenommen.
#
Beispiel:
baumgrep '-n strcat' '*.h' /usr/include/
```



```
Es werden alle System-Headerdateien nach dem String strcat
durchsucht. Zu allen gefundenen Strings wird noch die
Zeilennummer ausgegeben (Option -n), in der sie gefunden
wurden.
#
Autor: Egon ...
#

usage - gibt usage-Info aus
usage() {
 cat <<EOF >&2
 usage: baumgrep '[optionen] reg.ausdr' 'datei(en)' [directory(s)]
 optionen alle fuer grep zugelassenen Optionen
 reg.ausdr legt die zu suchenden Begriffe fest
 datei(en) legt die zu durchsuchenden Dateien fest;
 Expandierungszeichen sind dabei erlaubt.
 directorys geben die Wurzeln der zu durchsuchenden
 Directorybaeume an; ist kein directory angegeben,
 so wird Working-Directory angenommen.
 EOF
}

#--- main -----
if [$# -lt 2]
then
 usage
 exit 1
fi

regausdr=$1
dir=${3:-`pwd`}

for name in `find $dir -type d -print`
do
 grep $regausdr $name/$2
done
$ cx baumgrep
$ baumgrep '-n strncpy' '*.h' /usr/include
/usr/include/string.h: 11: *strncpy(),
$ baumgrep expr '*' /bin /usr/bin /etc
/bin/basename:exec /bin/expr \
/bin/dirname:exec /bin/expr \
/bin/lorder:# The first two expressions in the sed script
$ cp baumgrep ../bin
$
```

### 4.22.3 Prozesse mit bestimmten Namen löschen

Mit dem Kommando **kill** können Prozesse mit bestimmten Prozeßnummern (PID) entfernt werden. Allerdings hat dieses Löschen über die PID den Nachteil, daß zunächst mit **ps** die PID des entsprechenden Prozesses ermittelt werden muß. Mit dem nachfolgenden Shell-Skript *namkill* ist es nun möglich, Prozesse über die Angabe von Namen (Login-Name oder Prozeßname) zu löschen:

Als Optionen kennt dieses Shell-Skript:

**-l** die angegebenen Prozesse ohne Rückfrage löschen; Voreinstellung ist eine Rückfrage.

**-s signalnr** den entsprechenden Prozessen wird das Signal *signalnr* geschickt; Voreinstellung ist die Signalnummer 9.

```
$ cat namkill ↵
namkill Version v1.0 (17.1.1997)
loescht Prozesse mit bestimmten Namen
#
Syntax: namkill [-l] [-s signalnr] name
-l loescht die angegeb. Prozesse ohne Rueckfrage
(Voreinstellung ist Rueckfrage)
-s signalnr ruft kill mit signalnr auf
(Voreinstellung ist Signalnummer 9)
#
Autor: Egon ...
#

usage - gibt usage-Info aus
usage() {
 cat <<EOF >&2
 usage: namkill [-l] [-s signalnr] name
 -l loescht die angegeb. Prozesse ohne Rueckfrage
 (Voreinstellung ist Rueckfrage)
 -s signalnr ruft kill mit signalnr auf
 (Voreinstellung ist Signalnummer 9)
 EOF
}

#---- main -----
tmp_datei=/tmp/namkill.$$
SIG=9
trap 'rm -f $tmp_datei >/dev/null 2>/dev/null; exit 2' 0 1 2 3 15

while getopts ls: opt # Abarbeiten der Optionen
do
 case $opt in
 l) loesch=1;;
 s) SIG=$OPTARG;;
```

```

 \?) usage
 exit 1;;
 esac
done
shift `expr $OPTIND - 1`

if [$# -eq 0] # Mind. ein Prozessname muss angegeben sein
then
 echo "Mind. ein Prozessname muss angegeben sein" >&2
 usage
 exit 1
fi

args="$*"

ps -ef >$tmp_datei
cat $tmp_datei |
grep "$args" | # Alle entspr. Prozesse herausfiltern
while read zeile
do
 set $zeile # Pos.param. mit entspr. ps-Zeile setzen
 if [-n "$loesch"]
 then
 ps -p $2 >/dev/null
 if [$? -eq 0] # Prozess nur loeschen, wenn noch existent
 then
 kill -SIG $2
 fi
 else
 ps -p $2 >/dev/null
 if [$? -eq 0] # Prozess nur abfragen, wenn noch existent
 then
 echo "Loeschen: $zeile (j/n) : \c" >&2
 antw=`line </dev/tty`
 ps -p $2 >/dev/null
 if [\(" $antw" = "j" -o "$antw" = "J" \) -a $? -eq 0]
 then
 kill -SIG $2
 fi
 fi
 fi
done

rm -f $tmp_name
$ cx namkill ↵
$ cp namkill ../bin ↵
$
```

**Beispiel**

```
namkill -l egon
```

löscht alle Prozesse des Benutzers **egon** ohne Rückfragen.

```
namkill "find / -name"
```

zeigt alle Prozesse an, die mit dem Aufruf `find / -name ...` erzeugt wurden und fragt für jeden einzelnen Prozeß nach, ob dieser zu löschen ist. Falls der Benutzer diese Frage bejaht, so wird dieser beendet.

## 4.22.4 Terminalein- und -ausgaben auf anderen Terminals mitprotokollieren

Oft ist es für Benutzer (wie z.B. Kunden oder Service-Leute), die sich in anderen Gebäuden oder sogar anderen Städten befinden, schwierig, ihre Probleme am Telefon richtig zu verdeutlichen. In solchen Fällen wäre es nützlich, wenn diese Benutzer dem Experten ihre Probleme direkt am Bildschirm zeigen könnten. Dazu wurde das nachfolgende Shell-Skript *termkop* entwickelt, das zu allen auf der Kommandozeile angegebenen Login-Namen den entsprechenden Terminalnamen ermittelt (mit **who -T**) und im Anschluß alle Ein- und Ausgaben, die an diesem Terminal erfolgen, auf den Terminals der angegebenen Login-Namen mitprotokolliert.

```
$ cat termkop ↵
termkop Version v1.0 (5.2.1997)
zeigt die Ausgaben des eigenen Terminals auch
auf anderen Terminals an.
#
Syntax: termkop [loginname(n)]
#
Autor: Egon ...
#

#--- schicke ---
schicke() { # schickt den als $1 uebergebenen Text an entspr. Terminals
 i=0
 echo "$1"
 while ["$i" -lt "$argc"]
 do
 eval dev="$devnr${i}"
 echo "$1" >/dev/"$dev"
 i=`expr $i + 1`
 done
}

#----- main -----
trap 'schicke "termkop-Modus wurde verlassen";
 kill -9 $$ 1>&2 2>/dev/null' 1 2 3 15
```

```
if [$# -eq 0]
then
 echo "usage: $0 [loginname(n)]" >&2
 exit 1
fi

i=0
argv="$*"
argc="$#"

for b in $argv
do
 set "" `who -T|grep $b`
 if ["$2" != "$b" -o "$3" != "+"]
 then
 echo "$b ist entweder nicht angemeldet oder hat sein" >&2
 echo "Terminal fuer fremde Benutzer-Ausgaben gesperrt" >&2
 fehler=1
 else
 geraete="{geraete}|tee /dev/$4"
 eval devnr${i}="$4"
 i=`expr $i + 1`
 fi
done
if [-n "$fehler"]
then
 echo "Skript wird aufgrund der fehlenden Verbindungen verlassen" >&2
 exit 2
fi

cat <<EOF >&2
 Dein Terminal ist jetzt im "TERMKOP-Modus".
 Alle Ein- und Ausgaben deines Terminals werden auf
 alle angegebenen Geraeten mitprotokolliert.
 Der TERMKOP-Modus kann jederzeit mit exit verlassen werden.
 Waehrend des TERMKOP-Modus wird als Promptzeichen immer
 termkop> ausgegeben.
EOF

while ["$kdo" != "exit"]
do
 schicke "termkop> \c"
 read kdo
 if ["$kdo"]
 then
 set "" $kdo
 if ["$2" = "cd"]
 then
 eval "$kdo"
 eval echo \'$kdo\' "$geraete"
 else
```

```
eval echo \'$kdo\' "$geraete" >/dev/null
eval $kdo "$geraete"
fi
done

schicke "termkop-Modus wurde verlassen"
$ cx termkop
$ cp termkop ../bin
$
```

### 4.22.5 Ausgabe von UNIX-Directorybäumen in graphischer Darstellung

Das nachfolgende Shell-Skript *tree* gibt Directorybäume in graphischer Form aus. Dabei wird zusätzliche Information zu den einzelnen Dateien über ein eventuell an den Dateinamen angehängtes Kürzel gegeben:

*dateiname\** einfache Datei und ausführbar  
*dateiname/* Directory  
*dateiname/-* leeres Directory  
*dateiname/-r* nicht lesbares Directory (Inhalt nicht auflistbar)  
*dateiname/-x* nicht ausführbares Directory (kein **cd** möglich)

Die Aufrufsyntax für dieses Skript ist

```
tree [-d] [-r "regulärer-ausdruck"] [directory]
```

Die Optionen bedeuten dabei:

- d** nur Directories aus dem entsprechenden Directorybaum anzeigen; diese Option hat höhere Priorität als die Option **-r**.
- r** "*regulärer-ausdruck*"  
nur Directories und zusätzlich die Dateien anzeigen, für die der angegebene *regulärer-ausdruck* zutrifft.

Wird kein *directory* angegeben, so wird der Directorybaum des Working-Directorys angezeigt.

```
$ pwd
/user1/egon/shellueb
$ cat tree
tree Version v1.0 (7.2.1997)
gibt Directorybaum in graphischer Form aus
#
Syntax: tree [-d] [-r "reg.ausdr"] [directory]
-d nur Dir. auflisten
-r nur Dir. und Dateien, die durch reg. ausdr. abgedeckt
```

```
#
Ist kein directory angegeben, so wird working
directory angenommen
#
#
Autor: Egon ...
#

usage - gibt usage-Info aus
usage() {
 echo "$0 [-d] [-r \"reg.ausdr\"] [directory]" >&2
 echo " -d nur Dir. auflisten" >&2
 echo " -r nur Dir. und Dateien, die durch reg. ausdr. abgedeckt" >&2
}

baum - fuer die graphische Baum-Ausgabe verantwortlich
baum()
{
 if [! -x "$1"] # Nur wenn x-Recht fuer Directory gesetzt
 then # kann doerthin gewechselt werden.
 echo "-x\\c" # Ansonsten wird nur Directoryname mit
 exit # angehaengtem "-x" ausgegeben und betreff.
 fi # Subshell sofort wieder verlassen.
 cd $1

 set -f # Ausschalten der Dateinamenexpandierung

 # Setzen der Positionsparameter mit allen Namen des Directories;
 # Ist dies nicht moeglich, so wird nur Directoryname mit angehaengten
 # "-r" ausgegeben, da Dir. nicht lesbar. Auch in diesem Fall wird die
 # betreffende Subshell wieder verlassen.
 set "" `ls -a 2>/dev/null` || { echo "-r\\c"; exit; }

 shift # Um ersten leeren Positionsparameter zu beseitigen

 if [$# -eq 2] # Wenn nur 2 Positionsparameter vorhanden
 then # (. und ..), so ist Directory leer und es wird
 echo "-\\c" # nur Directoryname mit angehaengten "-"
 fi # ausgegeben.

 args="$*"
 argus=""

 # Abarbeiten der angegebenen Optionen
 # Verbleibende Dateien werden in Var. args gespeichert.
 if [-n "$OPTION"]
 then
 case $OPTION in
 *) for i in $args
 do
```

```

 [-d "$i"] && argus="$argus $i"
 done
 argus="$argus"
 argus="";;
esac
case $OPTION in
 *) for i in $args
 do
 if [-d "$i"] || case $i in
 $ra) ;;
 *) false;;
 esac
 then
 argus="$argus $i"
 fi
 done
 argus="$argus"
 argus="";;
esac
fi

Setzen der Positionsparameter mit den verbliebenen Dateien
if [-n "$args"]
then
 set "" `ls -adF $args 2>/dev/null`
else
 set ""
fi

shift # Um ersten leeren Positionsparameter zu beseitigen

Ausgabe des graphischen Baumteils
while [-n "$2"]
do
 if ["$1" != "/" -a "$1" != "../"]
 then
 echo "\n${striche}|----$1\c"
 if [-d "$1"]
 then
 (striche="${striche}| "; baum $1)
 fi
 fi
 shift
done

if [-n "$1" -a "$1" != "/" -a "$1" != "../"]
then
 if [-d "$1"]
 then
 echo "\n${striche}\`----$1\c"
 striche="${striche} "; baum $1
 fi
fi

```



```

 else
 echo "\n${striche}\`----$1\n${striche}\c"
 fi
 fi
}

#----- main -----
#-----

trap 'echo; exit' 0 1 2 3 15

while getopts dr: opt # Abarbeiten der angegebenen Optionen
do
 case $opt in
 d) OPTION="${OPTION}d";;
 r) OPTION="${OPTION}r";ra="$OPTARG";;
 \?) usage; exit 1;;
 esac
done
shift `expr $OPTIND - 1`

dir=${1:-`pwd`} # Bestimmen der Wurzel des auszugebenden Baums
if [! -d "$dir"]
then
 echo "$0: $dir ist kein Directory" >&2
 exit 1
fi
echo "$dir\c"

baum $dir
$ chmod u+x tree
$ tree -d /usr

/usr
|----adm/
| `----sa/
|----admin/
| `----menu/
| |----diskmgmt/
| | `----harddisk/
| |----filemgmt/
| | `----bupsched/
| |----machinemgmt/
| |----packagemgmt/
| | |----uucpmgmt/
| | `----vpixmgmt/
| |----softwaremgmt/
| |----syssetup/
| |----ttymgmt/
| `----usermgmt/

```

```
| |----modgroup/
| `----moduser/
|----bin/
|----include/
| `----sys/
| `----fs/
| `----nfs/
|
|.....
|.....
$ tree -r "s*" /usr/include ↵
/usr/include
|----scnhdr.h
|----search.h
|----setjmp.h
|----sgtty.h
|----signal.h
|----stand.h
|----stdio.h
|----storclass.h
|----string.h
|----stropts.h
|----strselect.h
|----symbol.h
|----syms.h
|----sys/
| |----fs/
| | |----nfs/
| | | |----svc.h
| | | `----svc_auth.h
| | |
| | |----s5dir.h
| | |----s5fblk.h
| | |----s5filsys.h
| | |----s5inode.h
| | |----s5macros.h
| | `----s5param.h
|
|.....
|.....
$ cp tree ../bin ↵
$
```



# 5 Die Korn-Shell

*Wirf Deinen Lendenschurz nicht fort, wenn Du ein neues Kleid bekommen hast!*

*Spruchwort aus dem Kongo*

Die Korn-Shell ist eine Weiterentwicklung der Bourne-Shell. Sie ist somit weitgehend aufwärtskompatibel zur Bourne-Shell. Das bedeutet, daß die Korn-Shell zum einen über alle Konstrukte der Bourne-Shell verfügt, die auch in ihrer Syntax der der Bourne-Shell entsprechen, zum anderen aber auch noch zusätzliche Mechanismen anbietet.

Dies hat zur Folge, daß alle bereits vorhandenen Bourne-Shellskripts ohne Änderung auch unter der Korn-Shell ablauffähig sind, aber umgekehrt, Korn-Shellskripts nicht unbedingt unter der Bourne-Shell lauffähig sein müssen, da sie eventuell von neuen Konstrukten Gebrauch machen, die der Bourne-Shell unbekannt sind.

Diese Einschränkung steht allerdings dem Umstieg von der Bourne- auf die Korn-Shell nicht im Wege, da nicht zu erwarten ist, daß eine solche Entscheidung zum Wechsel der Shell durch das entsprechende Projekt-Management wieder rückgängig gemacht wird, wenn die einzelnen SW-Entwickler einmal die Vorzüge der Korn- gegenüber der Bourne-Shell erkannt haben.

Da die Korn-Shell eine Obermenge der Bourne-Shell ist, werden hier nur die Neuheiten und Unterschiede der Korn-Shell zur Bourne-Shell vertieft vorgestellt. Ist eine genauere Beschreibung erwünscht, so muß im Kapitel 4 bei der Bourne-Shell nachgeschlagen werden.

## 5.1 Erweiterungen und Neuheiten der Korn-Shell

Die Korn-Shell hat sich die C-Shell, welche allerdings nicht aufwärtskompatibel zur Bourne-Shell ist, zum Vorbild genommen. Gegenüber der Bourne-Shell bietet die Korn-Shell folgende Neuheiten an:

### ► Editieren von Kommandozeilen

Ein sehr großer Vorteil der Korn-Shell ist, daß sie das Editieren einer Kommandozeile mit **vi**- oder **emacs**-Kommandos zuläßt. Somit müssen bei einer fehlerhaften Kommandozeile nicht alle Zeichen »rückwärts« bis zur Stelle, an der eine Änderung vorzunehmen ist, zunächst gelöscht und dann wieder neu eingegeben werden.

► **History-Mechanismus**

Die Korn-Shell merkt sich Kommandos, die zuvor eingegeben wurden, in einer sogenannten *History-Datei*. Auf diese Datei kann mittels **vi**- oder **emacs**-Kommandos oder unter Verwendung des Kommandos **fc** zugegriffen werden. Somit ist es möglich, früher gegebene Kommandozeilen zu editieren und erneut aufzurufen, ohne daß diese vollständig neu einzutippen sind. Die History-Datei bleibt auch nach Beendigung einer UNIX-Sitzung bestehen.

► **Alias-Mechanismus**

Es ist möglich, an ganze Kommandozeilen, die häufig benötigt werden, einen Kurznamen (auch *Alias* genannt) zu vergeben, und diese dann unter Angabe des Alias aufzurufen.

► **Job-Kontrolle**

In der Korn-Shell ist es möglich, Jobs (Programme) zu stoppen und ihre Ausführung vom Vorder- in den Hintergrund bzw. umgekehrt zu verlegen.

► **Erweitertes cd-Kommando**

Mit dem Aufruf

`cd -`

ist es möglich, zum vorherigen Working-Directory zurückzukehren, ohne daß der Directory-Name erneut einzugeben ist. Weiter ist es möglich, das Kommando **cd** durch eine eigene Funktion zu ersetzen, um es den persönlichen Anforderungen anzupassen.

► **Tilde-Expandierung**

Auf das Home-Directory eines beliebigen Benutzers kann durch die Angabe `~loginname` zugegriffen werden, ohne daß dazu der Pfadname dieses Directories bekannt sein muß.

► **Eigenes Menü-Kommando**

Die Implementierung von Menüs in Korn-Shellskripts wird durch ein eigenes Kommando (**select**) erleichtert. Die Korn-Shell paßt dann die Menü-Ausgabe automatisch der Bildschirmgröße des jeweiligen Terminaltyps an.

► **Eingebaute Arithmetik für ganze Zahlen**

Die Korn-Shell kann ganzzahlige Berechnungen vom Dual- bis zum 36-System durchführen, ohne daß dafür – wie in der Bourne-Shell – ein nicht builtin-Kommando (wie **expr** oder **bc**) aufzurufen ist.

► **Substring-Operatoren**

Die Korn-Shell kann Teilstrings aus den Werten von Shell-Variablen extrahieren.

► **Variablen-Attribute**

Shell-Variablen in der Korn-Shell verfügen über Attribute, die z.B. festlegen, ob es sich um eine String- oder Integer-Variable handelt. Über Änderung eines Attributs ist es z.B. auch möglich, den String-Wert einer Variablen von Groß- in Kleinbuchstaben umzuwandeln.

► **Arrays**

Eindimensionale Arrays von Zahlen oder Strings sind in der Korn-Shell erlaubt.

► **Funktionslokale Variablen**

In Funktionen können lokale Variablen definiert werden.

► **Kommunikation mit Hintergrund-Prozessen**

In der Korn-Shell ist möglich, mit einem oder mehreren Programmen, die im Hintergrund ablaufen, zu kommunizieren.

► **Bessere Unterstützung bei der Fehlersuche**

Die Debugging-Möglichkeiten der Korn-Shell sind gegenüber denen der Bourne-Shell verbessert wurden. Deshalb kann ein Fehler schneller lokalisiert und behoben werden.

► **Verbesserte Ablaufgeschwindigkeit**

Die Korn-Shell ist drei- bis zehnmal schneller als die Bourne-Shell; dies gilt allerdings nicht für den Start einer Subshell, da das Korn-Shellprogramm umfangreicher ist und wesentlich mehr Mechanismen zur Manipulation und Überprüfung der jeweiligen Shell-Umgebung anbietet. Dieser kleine Nachteil wird jedoch durch die Tatsache wettgemacht, daß die Korn-Shell nicht sooft von sich aus Subshells startet wie vergleichsweise die Bourne- oder C-Shell; so startet die Korn-Shell z.B. für die Abarbeitung von Schleifen keine eigene Subshell.

## 5.2 Das Starten der Korn-Shell

Welche Shell als Login-Shell nach dem Anmelden zu starten ist, kann der Systemadministrator durch einen Eintrag in der entsprechenden Benutzerzeile in der Datei `/etc/passwd` festlegen<sup>1</sup>. Ist die Korn-Shell nicht die Login-Shell, so kann sie mit ihrem Programmnamen **ksh** aufgerufen werden. Erhält man in diesem Fall die Meldung

*ksh: not found*

---

1. Siehe Kapitel 2.

muß nach ihr gesucht werden<sup>1</sup>. Üblicherweise befindet sich das ksh-Programm in `/bin`. Andere Unterbringungsmöglichkeiten könnten `/usr/bin`, `/usr/sbin`, `/usr/local` oder `/usr/add-on` sein. Jedenfalls sollte das entsprechende Directory in der Variablen **PATH** mit aufgenommen werden, damit die Korn-Shell von nun ab nur mit ihrem Programmnamen **ksh** und nicht nur mit ihrem absoluten Pfadnamen aufgerufen werden kann.

Wird die Korn-Shell gestartet, führt sie wie die Bourne-Shell zunächst die Datei `.profile` aus.

Danach sucht sie in ihrer Umgebung nach der Variablen **ENV**. Ist diese Variable vorhanden, so wird als nächstes die in **ENV** angegebene Datei ausgeführt. Diese Datei erstellt der jeweilige Benutzer selbst, um die Umgebung der Korn-Shell seinen eigenen Bedürfnissen anzupassen. Nachfolgend wird `$HOME/.ksh_env` als **ENV**-Datei (Environment-Datei) angenommen, deren Inhalt auf den nachfolgenden Seiten ständig erweitert wird.

Zunächst sollte allerdings die Variable **ENV** in `.profile` entsprechend gesetzt werden. Zudem sollte in `.profile` noch die Variable **SHELL** mit den Pfadnamen der **ksh** gesetzt werden, da viele Programme (wie z.B. **vi**) den Inhalt dieser Variablen verwenden, um festzulegen, welche Shell aufzurufen ist, wenn in diesen Programmen der Start einer Shell gefordert ist<sup>2</sup>:

```
$ cat $HOME/.profile
PATH=/bin:/usr/bin/:.
ENV=$HOME/.ksh_env
SHELL=/bin/ksh # Pfadname der ksh (evtl. aendern)
export PATH ENV SHELL
$
```

Für die nachfolgenden Beispiele wird das Directory `/user1/egon/kshueb` benutzt:

```
$ pwd
/user1/egon
$ mkdir kshueb
$ cd kshueb
$ pwd
/user1/egon/kshueb
$
```

- 
1. Hinweis: Die Korn-Shell **ksh** ist erst seit System V.4 ein fester Bestandteil des UNIX-Software-Pakets.
  2. Eventuell sollte die alte Datei `.profile` von der Bourne-Shell zuvor gesichert werden, um sie auch später noch zur Verfügung zu haben.

## 5.3 Metazeichen

In der Korn-Shell existieren dieselben Metazeichen wie in der Bourne-Shell. Allerdings sind neue Metazeichen hinzugekommen:

| Metazeichen                 | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>kdo</i> >   <i>datei</i> | wie bei > wird die Standardausgabe in <i>datei</i> umgelenkt. Im Unterschied zu > wird jedoch der alte Inhalt von <i>datei</i> in jedem Fall überschrieben, selbst wenn die Option <b>noclobber</b> eingeschaltet ist.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <i>kdo</i> <> <i>datei</i>  | <i>datei</i> zum Lesen und Schreiben eröffnen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>\$(kdos)</b>             | Neue alternative Form der Kommandosubstitution.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>\$(10), \${11}, ..</b>   | Zugriff auf die Werte der Positionsparameter <b>10, 11, ...</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>?(pattern)</b>           | deckt kein oder ein Vorkommen des angegebenen <i>pattern</i> ab.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>*(pattern)</b>           | deckt kein, ein oder mehrere Vorkommen des angegebenen <i>pattern</i> ab.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>+(pattern)</b>           | deckt ein oder mehrere Vorkommen von <i>pattern</i> ab.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>@(pattern)</b>           | deckt genau ein Vorkommen des angegebenen <i>pattern</i> ab.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>!(pattern)</b>           | deckt die Strings ab, die nicht durch das angegebene <i>pattern</i> abgedeckt werden.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>~</b>                    | Tilde-Expandierung für Worte, die mit ~ (engl. <i>tilde</i> ) beginnen.<br><br>Auf das Home-Directory eines beliebigen Benutzers kann durch die Angabe <i>~loginname</i> zugegriffen werden.<br><br>Eine andere Anwendung von ~ ist z.B. <b>cd ~-</b> (Tilde und Minus), was einen Wechsel zum vorherigen Working-Directory bewirkt.                                                                                                                                                                                                                                                                                                                                                                              |
| <b>((ausdr))</b>            | arithmetische Auswertung von <i>ausdr</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <i>kdo</i>   <b>&amp;</b>   | Das Metazeichen   <b>&amp;</b> bewirkt, daß das Kommando <i>kdo</i> (wie bei der Angabe von <b>&amp;</b> ) im Hintergrund (parallel) abläuft und die Elternshell nicht auf die Beendigung von <i>kdo</i> wartet. Anders als beim Metazeichen <b>&amp;</b> wird hier allerdings zusätzlich eine "Zweiwege-Pipe" eingerichtet, über die der Eltern- und Kindprozeß miteinander kommunizieren können. "Zweiwege-Pipe" bedeutet dabei, daß der Elternprozeß über diese Pipe in die Standardeingabe von <i>kdo</i> schreiben oder dessen Standardausgabe lesen kann. Dazu muß der Elternprozeß die beiden builtin-Kommandos <b>print</b> (ähnlich zu <b>echo</b> ) und <b>read</b> mit der Option <b>-p</b> verwenden. |

Tabelle 5.1: Neue Metazeichen in der Korn-Shell



**Beispiel**

```
$ pwd
/user1/egon/kshueb
$ cd ~emil
$ pwd
/user1/emil
$ cd ~~
$ pwd
/user1/egon/kshueb
$ a=4+5
$ echo $a
4+5
$ ((a=4+5))
$ echo $a
9
$ cat rechne
 # Dieses Skript wird als Koprozess gestartet
 #
g=0
while read zahlen # Lesen einer Zeile aus der Pipe
do
 set $zahlen # Setzen der Positionsparameter
 for i # Fuer jeden einzelnen
 do # Positionsparameter addieren: Aufruf
 g=`bc <<- EOF # des Rechners bc (siehe Kap 9)
 $g+$i
 EOF`
 done
 print $g # Zwischenergebnis in Pipe schreiben
done
$ cat add
rechne |& # Starten des Skripts rechne als Koprozess

while read zeile
do
 print -p "$zeile" # Gelesene Zeile ueber Pipe
 # an Koprozess leiten
 read -p gesamt # Lesen des neu berechneten
 # Zwischenergebnisses aus der
 # Pipe zum Koprozess
 echo "---> $gesamt" # Ausgabe des Zwischenergebnisses
done
$ chmod u+x add rechne
$ add
1 2 3 4
---> 10
20 35
---> 65
12 1024 16
---> 1117
12345678
---> 12346795
[Bereichs-Überlauf]
[Strg-D]
$
```

## 5.4 Einfache Kommandos, Pipelines und Listen

Wie bei der Bourne-Shell gilt:

*Listen* werden aus einer oder mehreren *Pipelines* gebildet.

*Pipelines* setzen sich aus einem oder mehreren *Kommandos* zusammen.

*Kommandos* sind unterteilt in

- ▶ einfache Kommandos
- ▶ geklammerte Kommandos und
- ▶ Programmiersprach-Konstrukte.

Neu zu den Programmiersprach-Konstrukten hinzugekommen sind in der Korn-Shell:

- ▶ **[[ *ausdr* ]]**
- ▶ **select**-Anweisung,
- ▶ **function** *funktionsname* { *liste* ; } und
- ▶ **time** *pipeline*

Diese neuen Kommandos werden an späterer Stelle ausführlich beschrieben.

### 5.4.1 Einfache Kommandos und exit-Werte

Die Definition für einfache Kommandos entspricht der Definition, wie sie für die Bourne-Shell gegeben wurde. Das gleiche gilt für den exit-Status von Kommandos. Beide sind ausführlich in Kapitel 4.2.1 beschrieben.

### 5.4.2 Pipelines

Eine *Pipeline* ist unter der Korn-Shell genauso definiert wie für die Bourne-Shell (siehe Kapitel 4.4.2)

### 5.4.3 Listen

Wie in der Bourne-Shell gilt: Eine *Liste* ist eine Folge von ein oder mehreren *Pipelines*, welche durch die Metazeichen

**;**, **&**, **&&** oder **|**

voneinander getrennt sind.

Am Ende einer Liste dürfen allerdings nicht nur wie in der Bourne-Shell die Metazeichen **&** und **;**, sondern auch das zuvor vorgestellte Metazeichen **|&** angegeben werden.

Die Vorrangsregeln der hier angegebenen Trennzeichen von Pipelines in einer Liste sind:

(; gleich & gleich | &) < (&& gleich ||)<sup>1</sup>

Wird in einer Liste das Symbol |& verwendet, ist der exit-Status dieser Liste 0 (erfolgreich).

### 5.4.4 Syntaxdiagramme zur Pipeline und Liste

Mit dem gegenüber der Bourne-Shell neu hinzugekommenen Symbol |& ergeben sich für die Korn-Shell die in Abbildung 5.1 gezeigten Syntaxdiagramme zu den zuvor vorgestellten Begriffen:

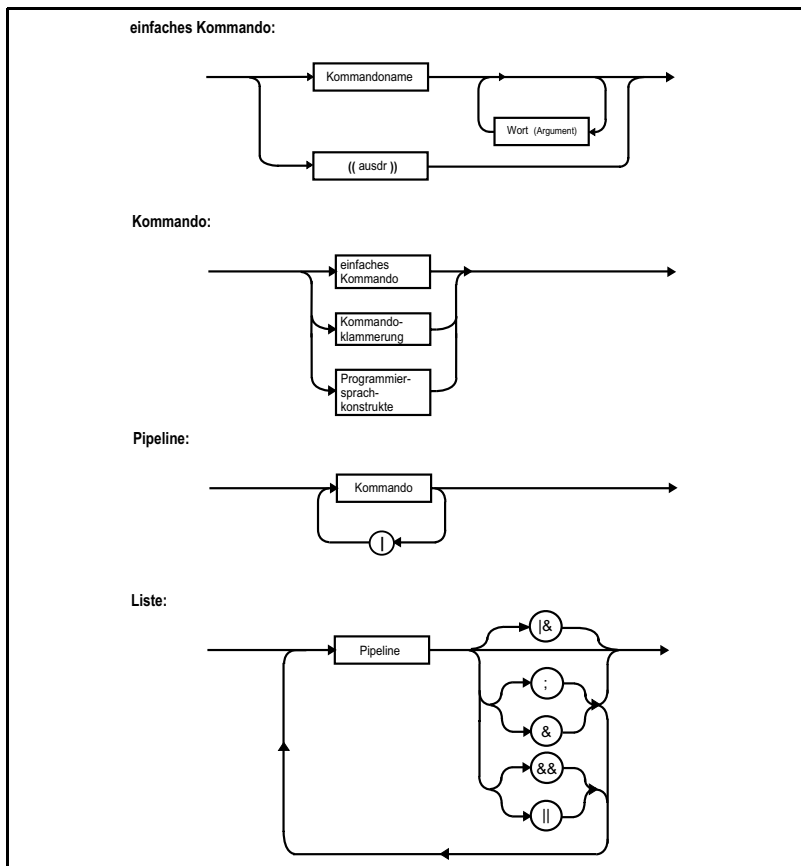


Abbildung 5.1: Syntaxdiagramme zu Kommandos, Pipelines und Listen

1. Die Bedeutung der Metazeichen ;, &, && und || kann in Kapitel 4.2.3 nachgeschlagen werden.

**Hinweis**

Wird eines der Symbole `|`, `&&` oder `||` an letzter Stelle einer Zeile angegeben, nimmt die **ksh** in diesem Fall an, daß die Kommandozeile noch nicht vollständig ist. Sie fordert den Benutzer durch Ausgabe des Sekundär-Promptstrings `>` zur weiteren Eingabe auf.

## 5.5 Kommentare

Wie in der Bourne-Shell gilt: Wenn ein Wort mit `#` beginnt, werden dieses Wort und alle nachfolgenden Zeichen dieser Zeile von der Korn-Shell als Kommentar interpretiert und als solcher von ihr ignoriert.

Auf manchen Systemen ist es möglich, mit der folgenden Angabe in der ersten Zeile eines Shell-Skripts:

```
#!/shell-pfadname
```

festzulegen, von welcher Shell dieses Skript auszuführen ist. So wäre es z.B. möglich, in der Bourne-Shell ein Skript von der Korn-Shell ausführen zu lassen:

```
$ pwd
/user1/egon/kshueb
$ cat homedir
#!/bin/ksh
cd ~egon # ksh-Anweisung: wechsele zu egon's home dir.
pwd
$ chmod u+x homedir
$ homedir
/user1/egon
$ pwd
/user1/egon/kshueb [Directory-Wechsel nur in Subshell stattgefunden]
$
```

## 5.6 Shell-Skripts (Shell-Prozeduren)

Wie in der Bourne-Shell können natürlich auch in der Korn-Shell Kommandos (bzw. Pipelines oder Listen) nicht nur interaktiv eingegeben werden, sondern in eine Datei geschrieben werden und diese Datei dann der **ksh** zur Abarbeitung der darin angegebenen Kommandos vorgelegt werden.

Solche Kommandodateien werden auch unter der Korn-Shell als *Shell-Skripts* (oder *Shell-Prozeduren*) bezeichnet.

Auch unter der Korn-Shell existieren mehrere Möglichkeiten, solche Shell-Skripts zu starten:

1. Aufruf mit **ksh**:

**ksh** *skript-name*

Wird ein Shell-Skript mit **ksh** aufgerufen, so wird eine neue Subshell<sup>1</sup> gestartet, die das angegebene Shell-Skript ausführt.

2. Alleinige Angabe des Shell-Skript-Namens (ohne **ksh**):

*skript-name*

Bei dieser Aufrufform muß die Datei, in welcher das Shell-Skript gespeichert ist, ausführbar sein, was beim Aufruf mit **ksh** nicht erforderlich ist.

Falls die entsprechende Shell-Skript-Datei nicht ausführbar ist, muß sie zuerst mit dem Kommando **chmod** dazu gemacht werden.

Wie beim Aufruf mit **ksh** wird auch hier eine eigene Subshell gestartet, welche die Ausführung des aufgerufenen Shell-Skripts übernimmt.

**Hinweis**

Auch hier gilt, daß als Dateinamen für Shell-Skripts keine Namen von UNIX-Kommandos gewählt werden sollten; die Gründe dafür wurden bereits in Kapitel 4.4 besprochen.

## 5.7 Kommandosubstitution

Wie in der Bourne-Shell gilt:

Kommandos, deren Standardausgabe von der **ksh** als Teil der Kommandozeile zu verwenden ist, müssen mit »Gegen-Apostroph« (engl.: *backquotes* oder *accents graves*):

``kommandos``

geklammert werden.

Wie in der Bourne-Shell, so gilt auch in der Korn-Shell:

- ▶ Alle Metazeichen behalten innerhalb einer Kommandosubstitution ihre Sonderbedeutung.
- ▶ Eine Schachtelung von Kommandosubstitutionen ist mit der Angabe von ``..\'`` (bzw. `\\`..\\``, usw.) möglich.
- ▶ Zu ``cat datei`` existiert eine äquivalente, schnellere Variante ``< datei``.

---

1. Kindprozeß zur aktuellen Shell.

Anders als in der Bourne-Shell wird bei einer Kommandosubstitution von built-in-Kommandos, die keine Ein- oder Ausgabeumlenkung verwenden, keine neue Subshell gestartet.

Da die Quoting-Regeln für diese Art der Kommandosubstitution (*'kommandos'*) sehr komplex sind<sup>1</sup>, bietet die Korn-Shell zusätzlich noch eine andere Variante von Kommandosubstitution an:

### **\$(kommandos)**

Bei dieser Form können die entsprechenden *kommandos* ohne Einschränkung durch bestimmte Quoting-Regeln genauso angegeben werden, wie wenn sie ohne Kommandosubstitution aufgerufen würden.

**Beispiel**

```
$ echo "Heute ist der $(date '+%d.%m.%y')"
```

Heute ist der 22.05.91

```
$ echo "Zur Zeit arbeiten $(who | wc -l) Benutzer am System"
```

Zur Zeit arbeiten 15 Benutzer am System

\$

Auch bei dieser neu hinzugekommenen Kommandosubstitution ist Schachtelung möglich.

**Beispiel**

```
$ pwd
```

/user1/egon/kshueb

```
$ cat addiere.c
```

```
#include <stdio.h>

main()
{
 float a, b;

 printf("Gib 2 Zahlen ein: ");
 scanf("%f %f", &a, &b);
 printf("Die Summe ist: %.3f\n", a+b);
}

$ cat cph
```

```
cp $(find /usr/include \
 -name $(grep '#include' addiere.c | line |
 cut -f2- -d' ' | tr -d '<>') \
 -print) .

$ ls
```

add  
addiere.c  
cph  
homedir  
rechne

```
$ chmod u+x cph
```

```
$ cph
```

---

1. Siehe Kapitel 4.8.2.

```
$ ls
add
addiere.c
cph
homedir
rechne
stdio.h
$
```

[ <---- neu im Directory ]

Die Wirkung der Kommandosubstitution **\$(cat datei)** kann auch hier durch **\$(<datei)** erreicht werden, wobei diese alternative Angabe wieder die schnellere Variante ist:

```
$ cat zaehle.txt
/etc/magic
/etc/passwd
/usr/include/s*.h
$ wc -w $(<zaehle.txt)
 946 /etc/magic
 46 /etc/passwd
 56 /usr/include/sccs.h
 502 /usr/include/scnhdr.h
 111 /usr/include/sd.h
 281 /usr/include/search.h
 309 /usr/include/setjmp.h
 293 /usr/include/sgtty.h
 236 /usr/include/shadow.h
 178 /usr/include/signal.h
 445 /usr/include/stand.h
 76 /usr/include/stdarg.h
 201 /usr/include/stddef.h
 856 /usr/include/stdio.h
 405 /usr/include/stdlib.h
 339 /usr/include/storclass.h
 309 /usr/include/string.h
 109 /usr/include/stropts.h
 280 /usr/include/strselect.h
 834 /usr/include/syms.h
6812 total
$
```

Werden innerhalb von **\$(..)** runde Klammern verwendet, so ist deren Sonderbe-  
deutung durch Voranstellen von **\** auszuschalten:

```
$ echo $(echo \(*\))
(*)
$
```

Es gibt jedoch eine Ausnahme zu beachten. Wird in **\$(..)** eine **case**-Anweisung verwendet, so ist vor jeder pattern-Angabe eine öffnende Klammer anzugeben, um sicherzustellen, daß gleich viele öffnende wie schließende Klammern in **\$(..)** vorhanden sind.

```
$ cat hexz ↵
echo "Gib Hexaziffer ein"
read hziff
echo "$hziff (16) = $(case $hziff in
 ([0-9]) echo $hziff;;
 ([Aa]) echo 10;;
 ([bB]) echo 11;;
 ([cC]) echo 12;;
 ([dD]) echo 13;;
 ([eE]) echo 14;;
 ([fF]) echo 15;;
 (*) echo "Keine Hexaziffer";;
esac) (10)"

$ chmod u+x hexz ↵
$ hexz ↵
Gib Hexaziffer ein
7 ↵
7 (16) = 7 (10)
$ hexz ↵
Gib Hexaziffer ein
b ↵
b (16) = 11 (10)
$ hexz ↵
Gib Hexaziffer ein
x ↵
x (16) = Keine Hexaziffer (10)
$
```

In **ksh**-Versionen, die vor dem 3.6.1986 freigegeben wurden, ist diese Angabe von **(** vor einem **case**-pattern allerdings nicht erlaubt; das wiederum bedeutet, daß in diesen **ksh**-Versionen keine **case**-Anweisung in **\$(..)** erlaubt ist.

## 5.8 Shell-Parameter

Die Korn-Shell kennt wie die Bourne-Shell zwei Arten von Parametern:

### Positionsparameter

Ihr Name wird als Ziffer 0, 1, 2,..., 9 angegeben. Wie später gezeigt wird, ist in der Korn-Shell auch ein Zugriff auf die Positionsparameter **10**, **11**, usw. mit der Angabe **\$(10)**, **\$(11)**, usw. möglich.



## Shell-Variablen

Ihr Name ist

- ▶ Ein Bezeichner (siehe Kapitel 3) oder
- ▶ eines der Zeichen `* @ # ? - $ !`

Wie bei der Bourne-Shell wird durch Voranstellen von `$` vor einem Parameternamen dessen Wert angesprochen:

`$parameter` entspricht: Wert von `parameter`

Anders als in der Bourne-Shell haben Bezeichner-Variablen nicht nur einen Wert, sondern können zusätzlich noch ein oder mehrere Attribute besitzen. Die gleichzeitige Zuweisung von Werten und Attributen an Bezeichner-Variablen kann mit dem neuen builtin-Kommando **typeset**<sup>1</sup> vorgenommen werden.

### 5.8.1 Positionsparameter

Für die Positionsparameter gelten weitgehend die gleichen Regeln wie in der Bourne-Shell:

Positionsparameter stellen wie in der Bourne-Shell die an ein Shell-Skript übergebenen Argumente zur Verfügung, wobei das 1. Argument dem Parameter 1, das 2. Argument dem Parameter 2, usw. zugewiesen wird. Dem Parameter 0 wird der Name des aufgerufenen Shell-Skripts zugewiesen.

Auf die Werte der einzelnen Parameternamen kann wieder durch Voranstellen des `$`-Zeichens zugegriffen werden.

#### Beispiel

```
$ cat ausgab[↵]
echo Das erste Argument ist $1
echo Das zweite Argument ist $2
echo Der Skriptname ist $0
$ chmod u+x ausgab[↵]
$ ausgab hans fritz franz[↵]
Das erste Argument ist hans
Das zweite Argument ist fritz
Der Skriptname ist ausgab
$
```

Mit dem builtin-Kommando **set** können den Positionsparametern auch wieder explizit Werte zugewiesen werden. Die beim Aufruf von **set** angegebenen Argumente werden dabei in der Reihenfolge ihrer Angabe den Positionsparametern zugewiesen.

---

1. **typeset** wird in Kapitel 5.8.3 vorgestellt.

Der Positionsparameter 0 wird durch den **set**-Aufruf nicht neu gesetzt, sondern behält weiterhin als Wert den Namen der Shell "ksh" bzw. des aufgerufenen Shell-Skripts<sup>1</sup>:

```
set argument1 argument2 argument3
 | | |
 0 1 2 3 Positionsparameter
 |
 "ksh" bzw.
 Skriptname
```

Werden innerhalb eines Shell-Skripts den Positionsparametern mit **set** neue Werte zugewiesen, so werden deren alte Inhalte (wie z.B. die Argumente aus der Kommandozeile) überschrieben.

Bei den zu **set** angegebenen Argumenten findet Dateinamen-Expansion statt.

Mit dem Kommando **shift** können die Werte der Positionsparameter wieder (nach links) verschoben werden.

Mit dem Aufruf

```
set --
```

werden alle Positionsparameter gelöscht:

```
$ set hans emil fritz ↵
$ echo $2 ↵
emil
$ echo $* ↵
hans emil fritz
$ set -- ↵
$ echo $2 ↵

$ echo $* ↵

$
```

Mit dem Aufruf

```
set -s
```

werden alle Positionsparameter nach dem ASCII-Code sortiert:

```
$ set hans emil fritz ↵
$ echo $2 ↵
emil
$ echo $* ↵
hans emil fritz
```

---

1. Dies läßt sich dadurch erklären, daß **set** kein eigenes Kommando, sondern ein builtin-Kommando (Programmteil) der **ksh** ist.

```
$ set -s
$ echo $2
fritz
$ echo $*
emil fritz hans
$
```

## 5.8.2 Shell-Variablen (Schlüsselwort-Parameter)

Der Name einer Korn-Shell-Variablen ist wie in der Bourne-Shell

- ▶ entweder als Bezeichner (siehe auch Kapitel 3)  
(Es gibt dabei keine Beschränkung bezüglich der Länge eines Bezeichners, und Groß- und Kleinbuchstaben werden unterschieden.)
- ▶ oder eines der Zeichen `* @ # ? - $ !`  
anzugeben.

Ebenso gilt die gleiche Syntax für die Zuweisung eines Wertes an eine Shell-Variable

`variablenname=wert [variablenname=wert]....1`

Nachfolgend wird wieder unterschieden zwischen:

- ▶ vom Benutzer frei wählbaren und
- ▶ von der **ksh** vordefinierten Variablennamen.

### Frei wählbare Variablennamen

Bezüglich der Wahl von benutzerdefinierten Variablennamen gelten die Regeln für Bezeichner (siehe Kapitel 3). Da die von der **ksh** vordefinierten Variablennamen immer aus drei oder mehr Großbuchstaben bestehen, ist es empfehlenswert, keine solchen Variablennamen zu wählen. So kann sichergestellt werden, daß eigene Variablen nicht mit Variablen kollidieren, die in späteren **ksh**-Versionen neu eingeführt werden.

Wie in der Bourne-Shell, so gilt auch in der Korn-Shell:

- ▶ Wird auf den Wert einer Variablen zugegriffen, bevor ihr explizit ein Wert zugewiesen wurde, dann liefert ein solcher Zugriff die leere Zeichenkette.
- ▶ Soll einer Shell-Variablen eine Zeichenkette zugewiesen werden, in der die Bedeutung aller Metazeichen, wie z.B. `$`, Leerzeichen usw. auszuschalten ist, dann ist diese Zeichenkette mit `'..'` zu klammern.
- ▶ Bei einer Zuweisung an eine Variable findet keine Dateinamen-Expansion statt.

---

1. Vor und nach dem Gleichheitszeichen darf dabei kein Trennzeichen angegeben sein.

- ▶ Die Definition einer Shell-Variablen kann mit dem Kommando **unset** wieder aufgehoben werden.
- ▶ Unter Verwendung des builtin-Kommandos **read** *variable(n)* kann eine Zeile von der Standardeingabe gelesen werden; dabei werden die einzelnen Wörter der Eingabezeile nacheinander den angegebenen Shell-Variablen *variable(n)* zugewiesen.

### Arrays

Neu in der Korn-Shell ist, daß sie auch eindimensionale Arrays mit bis zu 512<sup>1</sup> Elementen anbietet. Einem Element eines Arrays kann dabei unter Verwendung der folgenden Syntax ein Wert zugewiesen werden:

*arrayname[index]=wert*

Als *index* ist dabei jeder mögliche arithmetische Ausdruck<sup>2</sup> zugelassen, der einen Wert aus dem Bereich 0 bis 511 liefert.

Soll auf den Wert eines Arrayelements zugegriffen werden, so muß dies mit

***\${arrayname[index]}***

erfolgen.

Werden die Klammern ***{.}*** weggelassen:

***\$arrayname[index]***

so wird der Wert ***\$arrayname*** mit dem String konkateniert, der durch die Dateinamen-Expandierung von ***[index]*** geliefert wird.

Arrays müssen nicht wie in höheren Programmiersprachen deklariert werden, sondern werden beim ersten Zugriff auf ein Element automatisch von der **ksh** angelegt. Wenn jedoch die Größe eines Arrays bekannt ist und explizit festgelegt werden soll, so kann **typeset** verwendet werden.

Wird ein Arrayname alleine ohne Angabe von ***[index]*** verwendet,

*arrayname*

ist dies gleichbedeutend mit der Angabe

*arrayname[0]*

---

1. Implementations-abhängiger Wert; Manche **ksh**-Versionen lassen eventuell mehr Array-Elemente zu.  
2. Siehe Kapitel 5.8.3.

Die Angabe

`${arrayname[*]}` bzw.

`${arrayname[@]}`

liefert alle Array-Elemente (mit Leerzeichen getrennt).

Die Angabe

`${#arrayname[*]}` bzw.

`${#arrayname[@]}`

liefert die Anzahl der Elemente im Array *arrayname*.

#### Beispiel

```
$ word[1]=one
$ word[2]=two
$ word[3]=three
$ word[0]="zero ---> start"
$ echo ${word[3]} ${word[2]}
three two
$ echo ${word[1]}
one
$ echo ${word[0]}
zero ---> start
$ echo $word
zero ---> start
$ echo $word[1]
zero ---> start
$ echo $word[2]
zero ---> start
$ word="Lauf los"
$ echo ${word[0]}
Lauf los
$ echo ${word[*]}
Lauf los one two three
$ >abc
$ >abc2
$ ls abc*
abc
abc2
$ word[0]=abc
$ echo $word[2]
abc2
$ echo ${#word[*]}
4
$ word[10]=ten
$ echo ${#word[*]}
5
$ echo ${word[*]}
abc one two three ten
$
```

[Inhalt von word[0]]

[Inhalt von word[0] und pattern [1] ]  
["Lauf los" wird word[0] zugewiesen]

[Leere Datei abc anlegen]  
[Leere Datei abc2 anlegen]

[entspricht: echo abc[2] ]

Arrays können nicht exportiert werden.

Neu in der Korn-Shell ist weiterhin, daß Variablen nicht nur einen Wert, sondern auch Attribute besitzen, wie z.B. Integer-Variable, Funktionsname usw. Gleichzeitige Festlegung eines Werts und Attributs für eine Variable kann mit dem Kommando **typeset** erreicht werden<sup>1</sup>. Soll z.B. eine Variable als Integer-Variable gekennzeichnet werden, so kann dies mit

**typeset -i** *variable=wert*

erfolgen.

### Vordefinierte Shell-Variablen

Wie die Bourne-Shell, so bietet auch die Korn-Shell eine Reihe von Variablen an, deren Namen von ihr bereits fest vorgegeben sind. Bei diesen vordefinierten Variablen wird wieder unterschieden zwischen

- ▶ vom Benutzer veränderbaren Shell-Variablen
- ▶ Variablen, die ständig von der **ksh** automatisch gesetzt werden (auch automatische Variablen genannt).

### Vordefinierte, aber änderbare Shell-Variablen

| Variablenname  | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CDPATH</b>  | Wie in der Bourne-Shell enthält diese Variable die Suchpfade für das builtin-Kommando <b>cd</b> (siehe auch Kapitel 4.6.2).<br><i>Keine Voreinstellung.</i>                                                                                                                                                                                                                                                                                                                          |
| <b>COLUMNS</b> | Wenn diese Variable gesetzt ist, so legt ihr Wert die Weite des Editor-Fensters für die builtin-Editoren der <b>ksh</b> (siehe Kapitel 5.20) und für die Ausgaben von <b>select</b> -Listen (siehe Kapitel 5.16.7) fest.<br><i>Voreinstellung: COLUMNS=80.</i>                                                                                                                                                                                                                       |
| <b>EDITOR</b>  | Falls die Variable <b>VISUAL</b> nicht gesetzt ist und der Wert der Variablen <b>EDITOR</b> mit einem der Wörter <i>emacs</i> , <i>gmacs</i> oder <i>vi</i> endet, so wird der entsprechende builtin-Editor für die <b>ksh</b> eingeschaltet (siehe Kapitel 5.20).<br><i>Voreinstellung: EDITOR=/bin/ed.</i>                                                                                                                                                                         |
| <b>ENV</b>     | Der Inhalt dieser Variablen legt den Pfadnamen der Environment-Datei fest; die in dieser Datei enthaltenen Definitionen und Kommandos werden bei jedem Aufruf einer neuen <b>ksh</b> gelesen bzw. ausgeführt. Üblicherweise werden in der mit <b>ENV</b> spezifizierten Datei Funktions- und Variablen-Definitionen oder Alias-Vereinbarungen (siehe Kapitel 5.17) angegeben, so daß diese automatisch jeder Subshell zur Verfügung gestellt werden.<br><i>Keine Voreinstellung.</i> |

Tabelle 5.2: Vordefinierte, aber änderbare Shell-Variablen in der **ksh**

1. Siehe Kapitel 5.8.3.

| Variablenname   | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FCEDIT</b>   | enthält den Pfadnamen des Editors, der beim Kommando <b>fc</b> (siehe Kapitel 5.19.2) zu verwenden ist.<br><i>Voreinstellung: FCEDIT=/bin/ed.</i>                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>FPATH</b>    | enthält die Directories, welche die <b>ksh</b> nach "Funktionsdefinitions-Dateien" durchsuchen soll (siehe auch <b>autoload</b> in Kapitel 5.16.9). Wie bei <b>PATH</b> sind auch hier die einzelnen Directories mit Doppelpunkt (:) voneinander zu trennen.<br><br>Diese Variable ist nur in <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.<br><i>Keine Voreinstellung.</i>                                                                                                                                                                                     |
| <b>HISTFILE</b> | enthält den Pfadnamen einer Datei, in der die Kommando-History (Liste der zuletzt aufgerufenen Kommandozeilen) aufzubewahren ist, um zuvor eingegebene Kommandos eventuell an späterer Stelle wieder verwenden zu können (siehe Kapitel 5.19.1). Diese Variable wird bei jedem Aufruf der <b>ksh</b> ausgewertet, aber nicht während einer Sitzung.<br><br><i>Voreinstellung:</i> Falls diese Variable nicht gesetzt ist oder die darin angegebene Datei aufgrund von Zugriffsrechten nicht beschrieben werden darf, so wird als History-Datei die Datei <b>\$HOME/.sh_history</b> genommen. |
| <b>HISTSIZE</b> | Der Inhalt dieser Variablen legt fest, wie viele der zuletzt eingegebenen Kommandos aufzuheben sind, um sie eventuell später wieder zu verwenden. Diese Variable wird bei jedem Aufruf der <b>ksh</b> ausgewertet, aber nicht während einer Sitzung.<br><br><i>Voreinstellung: HISTSIZE=128.</i>                                                                                                                                                                                                                                                                                             |
| <b>HOME</b>     | Wie in Bourne-Shell: enthält Home-Directory des entsprechenden Benutzers.<br><br><i>Voreinstellung:</i> Wird beim Anmelden auf einen vom Systemadministrator (in <i>/etc/passwd</i> ) festgelegten Pfadnamen gesetzt.                                                                                                                                                                                                                                                                                                                                                                        |
| <b>IFS</b>      | ( <i>Internal Field Separators</i> ) enthält die Trennzeichen, welche zum Trennen einer Zeile in einzelne Wörter zu verwenden sind.<br><br>Anders als in der Bourne-Shell gelten die darin enthaltenen Trennzeichen nur für <b>read</b> , <b>set</b> und für das Resultat aus Parameter- und Kommandosubstitution.<br><br><i>Voreinstellung:</i> Leerzeichen, Tabulatorzeichen und Neuezeile-Zeichen.                                                                                                                                                                                        |

Tabelle 5.2: Vordefinierte, aber änderbare Shell-Variablen in der ksh (Fortsetzung)

| Variablenname    | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>LINES</b>     | <p>wird von <b>ksh</b> für die Ausgabe von <b>select</b>-Listen (siehe Kapitel 5.16.7) verwendet; ungefähr zwei Drittel der mit <b>LINES</b> festgelegten Zeilenzahl wird dabei ausgegeben. Neben <b>ksh</b> verwenden auch noch andere Programme diese Variable.</p> <p>Voreinstellung: <b>LINES=24</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>MAIL</b>      | Wie in Bourne-Shell: enthält den Pfadnamen der mailbox-Datei.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>MAILCHECK</b> | Wie in Bourne-Shell: legt die Zeitperiode (in Sekunden) fest, in der immer zu überprüfen ist, ob neue mail angekommen ist.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>MAILPATH</b>  | <p>Ist sehr ähnlich zur Bourne-Shellvariablen <b>MAILPATH</b>, außer daß ein <b>?</b> anstelle von <b>%</b> zu verwenden ist, um die zu einem mail-Pfadnamen gehörige Meldung von diesem abzugrenzen. Wird keine solche Meldung für einen Pfadnamen angegeben, so ist hier die voreingestellte Meldung "you have mail in \$_" (\$_ enthält den Pfadnamen der mailbox).</p> <p>Auch hier wird für die angegebene Meldung Parameter- und Kommandosubstitution durchgeführt. So würde z.B. die Zuweisung</p> <pre>MAILPATH="/usr/mail/egon:/usr/mail/gruppe?\$(logname), Postbote da"</pre> <p>bewirken, daß die Ankunft neuer mail in der mailbox <code>/usr/mail/egon</code> mit "you have mail in <code>/usr/mail/egon</code>" und die Ankunft neuer mail in der mailbox <code>/usr/mail/gruppe</code> mit "Login-Name, Postbote da" (z.B. "egon, Postbote da") gemeldet würde.</p> <p>Auch kann für die entsprechenden mailbox-Dateinamen die Kurzform <b>\$_</b> verwendet werden. So würde z.B. die Zuweisung</p> <pre>MAILPATH="/usr/mail/egon:/usr/mail/gruppe?Post in \$_ angekommen"</pre> <p>bewirken, daß die Ankunft neuer mail in der mailbox <code>/usr/mail/egon</code> mit "you have mail in <code>/usr/mail/egon</code>" und die Ankunft neuer mail in der mailbox <code>/usr/mail/gruppe</code> mit "Post in <code>/usr/mail/gruppe</code> angekommen" gemeldet würde.</p> <p>Keine Voreinstellung.</p> |
| <b>PATH</b>      | <p>Wie in Bourne-Shell: enthält die Suchpfade für Programme.</p> <p>Die Voreinstellung ist nicht wie in der Bourne-Shell:</p> <pre>PATH=:/bin:/usr/bin,</pre> <p>sondern</p> <pre>PATH=/bin:/usr/bin:<sup>a</sup></pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

Tabelle 5.2: Vordefinierte, aber änderbare Shell-Variablen in der ksh (Fortsetzung)



| Variablenname | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>PS1</b>    | <p>Wie in der Bourne-Shell enthält diese Variable den Primär-Promptstring, welchen die <b>ksh</b> ausgibt, wenn sie die Eingabe von Kommandos erwartet.</p> <p>Anders als in der Bourne-Shell kann allerdings der Promptstring nicht nur statisch, sondern auch dynamisch festgelegt werden, da die Parametersubstitution bei Klammerung mit <b>'.'</b> nicht bereits bei der Zuweisung, sondern immer nur beim Lesen von <b>PS1</b> ausgeführt wird.</p> <p>Ebenso neu gegenüber der Bourne-Shell ist, daß jedes Vorkommen von <b>!</b> im Promptstring bei der Ausgabe durch die entsprechende Kommandonummer ersetzt wird.</p> <p>Beispiel:</p> <pre>\$ PS1='!&gt;\$PWD: ' <sup>b</sup> 202&gt;/user1/egon/kshueb: cd /usr  203&gt;/usr: cd  204&gt;/user1/egon: cd kshueb  205&gt;/user1/egon/kshueb: PS1="\$ "  \$<sup>c</sup></pre> <p>Voreinstellung: <b>PS1="\$ "</b> (beim Superuser: <b>PS1="# "</b>)</p> |
| <b>PS2</b>    | <p>Wie in Bourne-Shell: enthält den Sekundär-Promptstring, welchen die <b>ksh</b> ausgibt, wenn sich eine Kommandozeile über mehrere Zeilen erstreckt, um anzuzeigen, daß sie noch auf weitere Eingaben wartet, bevor sie mit der Ausführung der gesamten Kommandozeile beginnt.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>PS3</b>    | <p>Enthält den Promptstring für die Auswahl beim Kommando <b>select</b> (siehe Kapitel 5.16.7)</p> <p>Voreinstellung: <b>PS3="#? "</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>PS4</b>    | <p>Enthält den Debugging-Promptstring. Der in <b>PS4</b> enthaltene Promptstring wird beim Debuggen von Shell-Skripts (Option <b>-x</b> gesetzt) auf die Standardfehlerausgabe ausgegeben (siehe Kapitel 5.21).</p> <p><b>PS4</b> ist nur in <b>ksh</b>-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.</p> <p>Voreinstellung: <b>PS4="+ "</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>SHACCT</b> | <p>Wie in Bourne-Shell: <b>ksh</b> schreibt in die über <b>SHACCT</b> festgelegte Datei Abrechnungsinformation.</p> <p>Keine Voreinstellung.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

Tabelle 5.2: Vordefinierte, aber änderbare Shell-Variablen in der ksh (Fortsetzung)

| Variablenname | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SHELL</b>  | <p>Wie in Bourne-Shell: enthält den Pfadnamen der Shell. Beim Aufruf einer neuen Shell wird geprüft, ob im Basisnamen dieses Pfadnamens ein <b>r</b> vorkommt. Wenn dies zutrifft, so wird diese <b>ksh</b> als eingeschränkte Shell (<i>restricted shell</i>) gestartet.</p> <p><i>Voreinstellung:</i> wird beim Anmelden auf einen vom Systemadministrator (in <i>/etc/passwd</i>) festgelegten Pfadnamen gesetzt.</p>                                                                                                                                                                                                                                                                                                                                                 |
| <b>TERM</b>   | <p>Wie in Bourne-Shell: spezifiziert das Terminal, an dem der Benutzer gerade arbeitet.</p> <p><i>Keine Voreinstellung.</i></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>TMOUT</b>  | <p>Wenn diese Variable einen Wert größer als 0 enthält, beendet sich die entsprechende <b>ksh</b> nach so vielen Sekunden, wenn nicht innerhalb dieser Zeitspanne ein neues Kommando eingegeben wird. Bevor sich die Shell allerdings beendet, gibt sie die Warnung "<i>shell time out in 60 seconds</i>" aus. Wenn dann nicht innerhalb von 60 Sekunden ein Kommando eingegeben oder zumindest die -Taste gedrückt wird, dann beendet sie sich.</p> <p>Die Korn-Shell kann so eingerichtet werden, daß es unmöglich ist, <b>TMOUT</b> einen Wert zuzuweisen, der größer als ein bestimmtes Maximum oder 0 ist.</p> <p><i>Voreinstellung: TMOUT=0</i> (bedeutet: unendliche Zeit)</p> |
| <b>VISUAL</b> | <p>Wenn der Wert dieser Variablen mit einem der Wörter <i>emacs</i>, <i>gmacs</i> oder <i>vi</i> endet, so wird der entsprechende builtin-Editor für die <b>ksh</b> eingestellt (siehe Kapitel 5.20)</p> <p><i>keine Voreinstellung.</i></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>TZ</b>     | <p>Wie in Bourne-Shell: legt die Zeitzone fest.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

Tabelle 5.2: Vordefinierte, aber änderbare Shell-Variablen in der ksh (Fortsetzung)

- Aus Sicherheitsgründen ist zuerst in */bin*, dann in */usr/bin* und zuletzt im Working-Directory zu suchen.
- Die automatische Variable **PWD** wird von der **ksh** immer mit dem Pfadnamen des Working-Directorys besetzt.
- Für die nachfolgenden Beispiele wird angenommen, daß die Variable **PS1** mit 

```
export PS1='!>$PWD: '
```

 in der Datei *.ksh\_env* (im Home-Directory) gesetzt wird.

Die momentanen Werte aller Shell-Variablen können mit den Kommandos **set** (ohne Angabe von Argumenten) oder **typeset** (ohne Angabe von Argumenten) am Bildschirm ausgegeben werden.

## Automatische Variablen

Die folgenden vordefinierten Variablen können niemals explizit durch den Benutzer gesetzt werden; sie werden ständig neu von der Korn-Shell gesetzt. Auf die Werte dieser automatischen Parameter kann wieder durch Voranstellen von **\$** zugegriffen werden:

| Variablenname | Bedeutung                                                                                                                                                                                                                            |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| #             | Anzahl der gesetzten Positionsparameter. Neben dem Aufruf eines Skripts oder einer Funktion wird <b>#</b> auch noch durch die Kommandos <b>set</b> , <b>shift</b> und <b>.</b> (Punkt-Kommando) neu gesetzt.                         |
| -             | (Minuszeichen) Optionen, welche beim Aufruf der Shell angegeben oder mit dem <b>set</b> -Kommando eingeschaltet wurden <sup>a</sup> .                                                                                                |
| ?             | Exit-Status des zuletzt im Vordergrund ausgeführten Kommandos.                                                                                                                                                                       |
| \$            | Prozeßnummer (PID) der aktuellen Shell.                                                                                                                                                                                              |
| !             | Die Prozeßnummer (PID) des zuletzt im Hintergrund gestarteten Kommandos.                                                                                                                                                             |
| *             | Alle Positionsparameter als <b>ein</b> String:<br>"\$*" entspricht "\$1 \$2 \$3 ..."<br>Als Trennzeichen für die einzelnen Parameter <b>\$1</b> , <b>\$2</b> , usw. wird dabei das erste in <b>IFS</b> angegebene Zeichen verwendet. |
| @             | Alle Positionsparameter als <b>einzelne</b> Strings:<br>"\$@" entspricht "\$1" "\$2" "\$3" ...                                                                                                                                       |

Tabelle 5.3: Automatische Variablen in der ksh

a. Siehe auch Kapitel 5.24.3.

### Beispiel

```
210>/user1/egon/kshueb: set "Hallo Franz" wie gehts
211>/user1/egon/kshueb: ALT_IFS="$IFS"
212>/user1/egon/kshueb: IFS=",$IFS"
213>/user1/egon/kshueb: for i in @$@
> do
> echo "$i"
> done
Hallo
Franz
wie
gehts
214>/user1/egon/kshueb: for i in "$@"
> do
> echo "$i"
> done
Hallo Franz
wie
```

```

gehts
215>/user1/egon/kshueb: for i in $*
> do
> echo "$i"
> done
Hallo
Franz
wie
gehts
216>/user1/egon/kshueb: echo "$*"
Hallo Franz,wie,gehts [Trennzeichen ist Komma (erstes Zeichen aus IFS)]
217>/user1/egon/kshueb: IFS="$ALT_IFS"
218>/user1/egon/kshueb: echo "$*"
Hallo Franz wie gehts [Trennzeichen ist Leerzeichen (erstes Zeichen aus IFS)]
219>/user1/egon/kshueb:

```

Den nachfolgenden automatischen Variablen kann der Benutzer zwar explizit Werte zuweisen, allerdings macht dies nur selten Sinn. Zum Beispiel enthält die Variable **PWD** immer das momentane Working-Directory. Wenn der Benutzer nun den Wert von **PWD** ändert, so bewirkt dies keinen Wechsel des Working-Directorys; zudem wird beim nächsten **cd**-Aufruf **PWD** wieder neu auf das entsprechende Working-Directory gesetzt.

| Variablenname | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <u>_</u>      | <p>(Unterstrich) Diese Variable hat mehrere Funktionen:</p> <ul style="list-style-type: none"> <li>– letztes Argument des vorherigen (Vordergrund-)Kommandos.</li> <li>– Beim Auswerten der Variablen <b>MAIL</b> enthält _ den Pfadnamen der mailbox.</li> <li>– Bei <b>ksh</b>-Versionen nach dem 3.6.1986 wird _ beim Aufruf eines Shell-Skripts mit dem Pfadnamen des Skripts gesetzt.</li> </ul> |
| <b>ERRNO</b>  | <p>Diese Integer-Variablen enthält immer die Fehlernummer des letzten fehlerhaften Systemaufrufs. Es ist zu beachten, daß die Fehlernummern systemabhängig sind.</p> <p>Mit der Zuweisung von 0 kann <b>ERRNO</b> zurückgesetzt werden.</p> <p><b>ERRNO</b> ist nur bei <b>ksh</b>-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.</p>                                                 |
| <b>LINENO</b> | <p>Die <b>ksh</b> setzt <b>LINENO</b> auf die momentane Zeilennummer in einem Skript oder in einer Funktion, bevor sie das entsprechende Kommando ausführt.</p> <p>Wenn <b>LINENO</b> explizit ein Wert zugewiesen wird, so wird damit ab der momentanen Zeile (nicht für die vorhergehenden) eine neue Zeilenzählung begonnen.</p>                                                                   |

Tabelle 5.4: Automatische ksh-Variablen, die änderbar sind

| Variablenname  | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | <b>LINENO</b> ist nur bei <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.                                                                                                                                                                                                                                                                                                                                                                              |
| <b>OLDPWD</b>  | enthält das vorherige Working-Directory.                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>OPTARG</b>  | Die <b>ksh</b> setzt die Variable <b>OPTARG</b> , wenn das Kommando <b>getopts</b> eine Option liest, die ein Argument verlangt; <b>OPTARG</b> wird dann das entsprechende Argument zugewiesen.<br><br><b>OPTARG</b> ist nur bei <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.                                                                                                                                                                       |
| <b>OPTIND</b>  | <b>getopts</b> setzt den Wert von <b>OPTIND</b> immer auf die Argument-Nummer der nächsten auszuwertenden Option.<br><br><b>OPTIND</b> wird immer automatisch auf 1 gesetzt, wenn <b>ksh</b> , ein Skript oder eine Funktion aufgerufen wird. Soll <b>getopts</b> eine neue Argumenten-Liste abarbeiten, so kann <b>OPTIND</b> explizit der Wert 1 zugewiesen werden.<br><br><b>OPTIND</b> ist nur bei <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden. |
| <b>PPID</b>    | Prozeßnummer der Elternshell ( <b>Parent Process ID</b> ).                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>PWD</b>     | momentanes Working-Directory.                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>RANDOM</b>  | <b>RANDOM</b> ist eine Integer-Variable, welcher die <b>ksh</b> bei jedem Zugriff eine Zufallszahl zwischen 0 und 32767 zuweist. Mit dem expliziten Zuweisen einer ganzen Zahl an <b>RANDOM</b> kann ein neuer Startwert für die zu generierende Zufallszahlenfolge festgelegt werden.                                                                                                                                                                                            |
| <b>REPLY</b>   | In dieser Variablen werden beim <b>select</b> -Kommando (siehe Kapitel 5.16.7) oder beim Aufruf des builtin-Kommandos <b>read</b> , wenn keine Argumente angegeben sind, die eingegebenen Zeichen gespeichert.                                                                                                                                                                                                                                                                    |
| <b>SECONDS</b> | Der Wert dieser Integer-Variablen enthält die seit dem Aufruf der <b>ksh</b> verstrichene Sekundenzahl. <b>SECONDS</b> kann auch explizit ein neuer Wert zugewiesen werden.                                                                                                                                                                                                                                                                                                       |

Tabelle 5.4: Automatische ksh-Variablen, die änderbar sind (Fortsetzung)

**Beispiel**

```
220>/user1/egon/kshueb: echo "Hallo" >/bin/ls
ksh: /bin/ls: cannot create
221>/user1/egon/kshueb: echo $
Hallo
222>/user1/egon/kshueb: echo $ERRNO
10 [Fehlernummer vom ersten echo-Kommando;systemabhaengig]
223>/user1/egon/kshueb: cat zufall
RANDOM=$$ # Start fuer Zufallszahlengenerator
echo "1.Zufallszahl: $RANDOM"
echo "2.Zufallszahl: $RANDOM"
```

```
echo "Zufzahl aus [1,100]: $(expr $RANDOM % 100 + 1)"
224>/user1/egon/kshueb: chmod u+x zufall
225>/user1/egon/kshueb: zufall
1.Zufallszahl: 677
2.Zufallszahl: 19145
Zufzahl aus [1,100]: 84
226>/user1/egon/kshueb: zufall
1.Zufallszahl: 1585
2.Zufallszahl: 21844
Zufzahl aus [1,100]: 36
227>/user1/egon/kshueb: echo $SECONDS
130
228>/user1/egon/kshueb: echo $SECONDS
139
229>/user1/egon/kshueb: SECONDS=50
230>/user1/egon/kshueb: echo $SECONDS; sleep 10; echo $SECONDS
63
73
231>/user1/egon/kshueb: echo $$
1286
232>/user1/egon/kshueb: echo $PPID
1256
233>/user1/egon/kshueb: ksh
234>/user1/egon/kshueb: echo $$
1359
235>/user1/egon/kshueb: echo $PPID
1286
236>/user1/egon/kshueb: exit
237>/user1/egon/kshueb: echo $PWD
/user1/egon/kshueb
238>/user1/egon/kshueb: cd /usr
239>/usr: echo $PWD
/usr
240>/usr: echo $OLDPWD
/user1/egon/kshueb
241>/usr: cd $OLDPWD
242>/user1/egon/kshueb: cat eingabe
echo "Gib was ein:"
read
echo "Du hast ---$REPLY--- eingegeben"
243>/user1/egon/kshueb: chmod u+x eingabe
244>/user1/egon/kshueb: eingabe
Gib was ein:
Das ist nur ein Test
Du hast ---Das ist nur ein Test--- eingegeben
245>/user1/egon/kshueb: set
CDSPELL=cdsPELL
ENV=/user1/egon/.ksh_env
ERRNO=10
FCEDIT=/bin/ed
HOME=/user1/egon
```

```
HZ=60
IFS=

LINENO=1
LOGNAME=egon
MAIL=/usr/spool/mail/egon
MAILCHECK=600
OLDPWD=/usr
OPTIND=0
PATH=/bin:/usr/bin:.
PPID=1256
PS1=!>$PWD:
PS2=>
PS3=#?
PS4=+
PWD=/user1/egon/kshueb
RANDOM=3756
SECONDS=279
SHELL=/bin/ksh
TERM=ansi
TMOUT=0
TZ=MET-1MESZ,M3.5.0,M9.5.0/03
_=eingabe
246>/user1/egon/kshueb:
```

### 5.8.3 Attribute von Variablen

Neu in der Korn-Shell ist, daß Variablen Attribute besitzen. Jeder Variablen (Bezeichner-Variablen) können ein oder mehrere Attribute zugeordnet werden. Wird das Attribut einer Variablen geändert, so wird auch der Wert dieser Variablen dem neuen Attribut entsprechend angepaßt. Um Attribute von Variablen ein- oder auszuschalten, steht das builtin-Kommando **typeset** zur Verfügung:

**typeset** [**±flrtuxH**] [**±iLRZ[n]**] [*variable*=[*wert* ..]]

Wird **typeset** alleine ohne Argumente aufgerufen, so werden alle momentan definierten Variablen mit ihren Attributen angezeigt.

Werden beim Aufruf von **typeset** nur Optionen angegeben, so werden alle Variablen, welche die mit den Optionen spezifizierten Attributen besitzen, aufgelistet:

Wurde vor den Optionen - (Minus) angegeben, so werden die betreffenden Variablen mit ihren derzeitigen Werten ausgegeben.

Ist vor den Optionen ein + (Plus) angegeben, so werden nur die entsprechenden Variablennamen ausgegeben.

Enthält ein **typeset**-Aufruf zumindest eine *variable*, dann werden allen erwähnten *variable(n)* die über die Optionen spezifizierten Attribute zugeordnet. Ein - (Minus) vor einer Option schaltet dabei diese ein und ein + (Plus) davor schaltet sie aus.

Mit **typeset** sind auch Variablenzuweisungen möglich; dabei sind mehrere Zuweisungen mit einem **typeset**-Aufruf möglich. Die Angabe von Optionen schaltet dabei wieder die entsprechenden Attribute für die *variable(n)* ein (-) bzw. aus (+).

Wenn **typeset** innerhalb einer Funktion aufgerufen wird, dann wird mit diesem Aufruf eine funktionslokale *variable* kreiert.

## Optionen

Folgende Optionen kennt **typeset**:

| Optionen      | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-u</b>     | ( <i>uppercase</i> ) alle Kleinbuchstaben werden in Großbuchstaben umgewandelt; schaltet die Option <b>-l</b> aus.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>-l</b>     | ( <i>lowercase</i> ) alle Großbuchstaben werden in Kleinbuchstaben umgewandelt; schaltet die Option <b>-u</b> aus.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>-i[n]</b>  | ( <i>integer</i> ) entsprechende Variable wird als Integer-Variable definiert. Ist eine ganze Zahl <i>n</i> angegeben, legt diese die Basis des Zahlensystems für diese Variable fest, andernfalls wird als Basis 10 angenommen. Bei jeder Zuweisung eines Werts an eine Integer-Variable wird dieser Wert als arithmetischer Ausdruck (nicht als String) interpretiert. Anstelle von<br><br><b>typeset -i variable=wert ..</b><br><br>kann auch das vordefinierte Alias <b>integer</b> (siehe Kapitel 5.17) verwendet werden:<br><br><b>integer variable=wert ..</b>                                                                                                            |
| <b>-L[n]</b>  | ( <i>Left justified</i> ) justiert den Wert der entsprechenden Variablen linksbündig und entfernt eventuell führende Leerzeichen. Ist eine ganze Zahl <i>n</i> (ungleich 0) angegeben, so legt diese die Anzahl der auszugebenden Zeichen fest. Ist kein <i>n</i> angegeben, wird diese Zeichenzahl durch die erste Zuweisung festgelegt.<br><br>Enthält ein zugewiesener Wert weniger Zeichen als für eine Variable festgelegt sind, werden bei der Ausgabe entsprechend viel Leerzeichen rechts angehängt; enthält dagegen ein zugewiesener Wert mehr Zeichen, so werden die rechts »überhängenden« Zeichen abgeschnitten.<br><br><b>-L</b> schaltet die Option <b>-R</b> aus. |
| <b>-LZ[n]</b> | ( <i>strip Leading Zeros</i> ) entspricht weitgehend der Option <b>-L[n]</b> . Allerdings werden bei diesem Attribut führende Nullen entfernt.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>-R[n]</b>  | ( <i>Right justified</i> ) justiert den Wert der entsprechenden Variablen rechtsbündig und entfernt Leerzeichen am Ende. Ist eine ganze Zahl <i>n</i> (ungleich 0) angegeben, legt diese die Anzahl der auszugebenden Zeichen fest. Ist kein <i>n</i> angegeben, wird diese Zeichenzahl durch die erste Zuweisung festgelegt.                                                                                                                                                                                                                                                                                                                                                    |



| Optionen                        | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                 | <p>Enthält ein zugewiesener Wert weniger Zeichen als für eine Variable festgelegt sind, werden bei der Ausgabe entsprechend viele Leerzeichen davor ausgegeben; enthält dagegen ein zugewiesener Wert mehr Zeichen, werden die links »überhängenden« Zeichen abgeschnitten.</p> <p><b>-R</b> schaltet die Option <b>-L</b> aus.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>-Z[n]</b> oder <b>-RZ[n]</b> | <p>(<b>Right Zero filled</b>) entspricht weitgehend der Option <b>-R[n]</b>. Allerdings wird hier der Wert bei der Ausgabe links mit führenden Nullen aufgefüllt. Dies geschieht allerdings nur dann, wenn das erste »echte« Zeichen eine Ziffer ist, ansonsten wird der Wert mit entsprechend viel führenden Leerzeichen ausgegeben.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>-r</b>                       | <p>(<b>readonly</b>) markiert Variablen als »nur lesbar«. Der Versuch, den Wert einer solchen Variablen zu ändern, resultiert in einer Fehlermeldung. Der Inhalt einer »nur lesbaren« Variablen kann nur dann geändert werden, wenn diese Variable entweder mit <b>unset</b> gelöscht oder aber mit <b>typeset +r</b> das Attribut »nur lesbar« ausgeschaltet wird. Das Attribut »nur lesbar« kann auch mit dem builtin-Kommando <b>readonly</b> eingeschaltet werden; allerdings bestehen dabei doch gewisse Unterschiede:</p> <p>Während eine mit <b>typeset -r</b> eingeführte Variable wieder mit <b>unset</b> entfernt werden kann, ist dies bei <b>readonly</b> nicht möglich.</p> <p>Innerhalb einer Funktion wird mit <b>typeset -r</b> eine funktionslokale Variable definiert; bei <b>readonly</b> dagegen eine globale Variable.</p> |
| <b>-x</b>                       | <p>(<b>export</b>) markiert Variablen für den Export an Subshells. Die <b>ksh</b> setzt dieses Attribut für alle Variablen, die sie von der Elternshell geerbt hat. Wird dann der Wert einer so geerbten Variablen geändert, wird dieser neue Wert (nicht der Wert von der Elternshell) an alle Subshells vererbt. Dies ist ein Unterschied zur Bourne-Shell, wo jede Variable neu exportiert werden muß, wenn ein geänderter Wert (nicht der Wert aus der Elternshell) an weitere Subshells zu vererben ist.</p> <p>Eine Variable kann neben <b>typeset -x</b> auch noch mit dem builtin-Kommando <b>export</b> exportiert werden. Innerhalb einer Funktion wird allerdings mit <b>typeset -x</b> eine funktionslokale Variable definiert, während <b>export</b> dagegen eine globale Variable einführt.</p>                                   |
| <b>-H</b>                       | <p>(<b>Host operating system pathname mapping</b>) nur bei Nicht-UNIX-Systemen anwendbar. <b>ksh</b> ignoriert diese Option auf UNIX-Systemen. Bei jedem Zugriff auf eine Variable mit diesem Attribut, ändert sie den Wert (Pfadnamen) dieser Variablen so, daß er den Pfadnamen-Regeln des jeweiligen Nicht-UNIX-Systems entspricht.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>-t</b>                       | <p>(<b>tagged</b>) markiert Variablen für benutzereigene Zwecke. Dieses Attribut wird von der <b>ksh</b> nicht benutzt.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

Die Option **-f** wird an späterer Stelle besprochen.

```
Beispiel 195>/user1/egon/kshueb: PS1="$ "
$ typeset -u a="hallo"
$ echo $a
HALLO
$ typeset -l a
$ echo $a
hallo
$ read a
Das ist ein TEST
$ echo $a
das ist ein test
$ integer x=6
$ typeset -i16 y=5*x
$ echo $y
16#1e [1e (hexadezimal); entspricht 30 im Zehnersystem]
$ typeset -i z=$y
$ echo $z
16#1e
$ typeset -i2 d
$ d=$y
$ echo $d
2#11110
$ typeset -R10 vv="Vorname" nn="Nachname" ww="Wohnort"
$ typeset -R10 vor="Egon" nach="Mueller"
$ wohnort="90480 Nuernberg"
$ typeset -R10 wohnort
$ echo "vvnn$ww\n$vor$nach$wohntort"
Vorname Nachname Wohnort
Egon Mueller Nuernberg
$ typeset -L10 vv nn ww vor nach wohnort
$ echo "vvnn$ww\n$vor$nach$wohntort"
Vorname Nachname Wohnort
Egon Mueller Nuernberg
$ typeset -u vv nn ww vor nach wohnort
$ echo "vvnn$ww\n$vor$nach$wohntort"
VORNAME NACHNAME WOHNORT
EGON MUELLER NUERNBERG
$ unset x y
$ typeset -L3 x="abcd" y
$ y=3
$ echo "$y:$x"
3 :abc
$ typeset -LZ3 x y=0003
$ echo "$y:$x"
:abc
$ typeset -R3 x y
$ y=3
$ echo "$y:$x"
3:bcd
$ typeset -Z3 x y
$ x="abcd"
```

```
$ echo "$y:$x"
003:bcd
$ typeset -l [Zeige alle Var. mit Werte, die Attribut -l haben]
a=das ist ein test
$ typeset +l [Zeige alle Var. ohne Werte, die Attribut -l haben]
a
$ typeset -t HOME mdir=PWD mzeit=$(date +%T)
$ typeset +t [Zeige alle mit -t markierten Var. ohne Wert]
HOME
mdir
mzeit
$ typeset -t [Zeige alle mit -t markierten Var. mit Wert]
HOME=/user1/egon
mdir=/user1/egon/kshueb
mzeit=14:20:38
$ integer i=2#1010101001
$ echo $i
2#1010101001
$ i=1023
$ echo $i
111111111
$ ((i=16#a4f+16#bad)) [Addition zweier Hexzahlen]
$ echo $i
2#1010111111100
$ typeset [Zeige alle Shell-Variablen mit ihren Attributen]
uppercase nn
export HZ
uppercase vor
export PATH
integer ERRNO
integer OPTIND
function LINENO
export LOGNAME
export MAIL
function SECONDS
readonly integer PPID
export PS1
PS3
PS2
OPTARG
function RANDOM
uppercase ww
export SHELL
tagged mdir
integer TMOUT
uppercase vv
uppercase nach
export tagged HOME
export _
lowercase a
integer base 2 d
```

```
FCEDIT
export TERM
integer base 2 i
export PWD
tagged mzeit
export TZ
export ENV
zerofill 3 rightjust 3 x
zerofill 3 rightjust 3 y
integer base 16 z
uppercase wohnort
integer MAILCHECK
$
```

Um beispielsweise von einer eingegebenen Antwort nur das erste Zeichen gelten zu lassen, könnte folgendes angegeben werden:

```
typeset -lLl antwort
echo "Gib j oder n ein:"
read antwort
if ["$antwort" = "j"]
then
 echo "Zugestimmt"
else
 echo "Abgelehnt"
fi
```

**typeset** kann auch verwendet werden, um Arrays mit einer festen Anzahl von Elementen zu definieren. So legt z.B. die Deklaration

**typeset x[100]**

ein Array mit 101 Elementen (Indizes von 0 bis 100) fest.

#### Beispiel

```
255>/user1/egon/kshueb: cat lotto ↵
RANDOM=$$ # Startwert fuer Zufallszahlengenerator

typeset -i zahl[49] i=1 z

while [$i -le 6] # 6 Zahlen sind zu ziehen
do
 ((z=$RANDOM % 49 + 1)) # Zahl zwischen 1 und 49
 while [zahl[$z] -eq 1] # wenn Zahl schon gezogen,
 do ((z=$RANDOM % 49 + 1)) # dann neuer Versuch
 done
 echo $z # Zahl ausgeben
 zahl[$z]=1 # Merken, dass Zahl bereits gezogen
 ((i=i+1)) # i inkrementieren
done
256>/user1/egon/kshueb: chmod u+x lotto ↵
257>/user1/egon/kshueb: lotto ↵
41
```

```
14
13
6
16
20
258>/user1/egon/kshueb: lotto ↵
23
33
41
10
44
40
259>/user1/egon/kshueb:
```

Liegt keine solche Deklaration für ein Array vor, so wird beim ersten Zugriff auf ein Arrayelement ein Array mit der maximalen Größe von 512<sup>1</sup> Elementen angelegt.

### 5.8.4 Spezielle Variablenausdrücke

Wie die Bourne-Shell unterscheidet auch die Korn-Shell zwischen undefinierten Variablen und Variablen, welchen explizit der »Nullwert« (leere Zeichenkette) zugewiesen wurde.

Neben dem einfachen Zugriff auf den Wert einer Variable mit

#### **`$variable`**

bietet auch die Korn-Shell einige spezielle Zugriffsmöglichkeiten auf die Werte von Variablen:

#### **`${variable}`**

ist wie in der Bourne-Shell identisch zur Angabe **`$variable`**.

Ein Variablenname muß immer dann in `{..}` angegeben werden, wenn

- ▶ dem Variablennamen ein Buchstabe, eine Ziffer oder ein Unterstrich folgt.
- ▶ ein Zugriff auf ein Arrayelement erfolgen soll:

#### **`${arrayname[index]}`**

- ▶ auf Positionsparameter mit mehr als einer Ziffer (größer als 9) zuzugreifen ist:

```
259>/user1/egon/kshueb: set a b c d e f g h i j k l m n o p q ↵
260>/user1/egon/kshueb: echo $12 ↵
a2
261>/user1/egon/kshueb: echo ${12} ↵
1
```

---

1. Systemabhängig

```
262>/user1/egon/kshueb: eval echo \${$#} ↵
q
263>/user1/egon/kshueb:
```

Die folgenden Ausdrücke sind identisch zur Bourne-Shell<sup>1</sup>:

### Verwendung von default-Werten

```
${variable:-wort}
${variable-wort}
```

### Zuweisen von default-Werten

```
${variable:=wort}
${variable=wort}
```

### Fehlermeldung, wenn Nullwert oder nicht gesetzt

```
${variable:?wort}
${variable?wort}
```

### Verwendung eines alternativen Werts

```
${variable:+wort}
${variable+wort}
```

Neu in der **ksh** hinzugekommen sind die folgenden Variablenausdrücke:

### Länge eines Strings

```
${#variable}
```

liefert die Länge des Strings, der in *variable* gespeichert ist. Wird für *variable* \* oder @ angegeben, so liefert dieser Ausdruck die Anzahl der Positionsparameter.

**Beispiel**

```
$ PS1="$ " ↵2
$ satz="Wie lang bin ich denn ?" ↵
$ echo ${#satz} ↵
23
$ echo $HOME ↵
/user1/egon
$ echo ${#HOME} ↵
11
$ set one two three four five six ↵
$ echo ${#*} ↵
6
$ shift 2 ↵
```

- 
1. Siehe Kapitel 4.6.3.
  2. Zur besseren Lesbarkeit wird für die nächsten Beispiele der Primärprompt immer mit **PS1="\$ "** gesetzt.

```
$ echo ${#@} ↵
4
$ namen="*" ↵
$ echo "$namen" ↵
*
$ echo ${#namen} ↵
1
$ echo $namen ↵
abc abc2 add addiere.c ausgab cph eingabe hexz homedir lotto rechne stdio.h
zaehle.txt zufall
$ a=`echo $namen` ↵
$ echo ${#a} ↵
93
$
```

### Anzahl der Elemente eines Arrays

`${#variable[*]}` oder  
`${#variable[@]}`

liefert die Anzahl der Elemente des Arrays *variable*.

#### Beispiel

```
$ x[1]=4 x[3]=17 x[6]="Hallo Egon" ↵
$ echo ${#x[*]} ↵
3
$ x[12]=144 ↵
$ echo ${#x[*]} ↵
4
$ typeset -u name[100] ↵
$ echo ${#name[*]} ↵
0
$ name[10]="Meier" ↵
$ echo ${#name[@]} ↵
1
$
```

### Entfernen eines kleinen linken Teilstrings

`${variable#pattern}`

Wenn der über *pattern*<sup>1</sup> angegebene reguläre Ausdruck den Anfang des in *variable* gespeicherten Strings abdeckt, liefert dieser Ausdruck den Wert von *variable* ohne den **kleinstmöglichen** durch *pattern* abgedeckten linken Teilstring, andernfalls liefert dieser Ausdruck den vollständigen Wert von *variable*.

---

1. Siehe Dateinamen-Expandierung in Kapitel 5.9.

**Beispiel**

```
$ echo $HOME
/user1/egon
$ echo $PWD
/user1/egon/kshueb
$ echo ${PWD/#$HOME/}
kshueb
$ text=Wasserfrosch
$ echo ${text/#*s}
serfrosch
$ echo ${text/??}
sserfrosch
$ echo ${text/#Wasser}
frosch
$ echo ${text/#frosch}
Wasserfrosch
$ echo ${text/#Waser}
Wasserfrosch
$
```

### Entfernen eines großen linken Teilstrings

**`${variable##pattern}`**

Wenn der über *pattern*<sup>1</sup> angegebene reguläre Ausdruck den Anfang des in *variable* gespeicherten Strings abdeckt, liefert dieser Ausdruck den Wert von *variable* ohne den **größtmöglichen** durch *pattern* abgedeckten linken Teilstring, andernfalls liefert dieser Ausdruck den vollständigen Wert von *variable*.

**Beispiel**

```
$ echo $PWD
/user1/egon/kshueb
$ echo ${PWD/#*/}
kshueb
$ echo ${PWD/**/}
user1/egon/kshueb
$ text=Wasserfrosch
$ echo ${text/#*s}
ch
$ echo ${text/#frosch}
Wasserfrosch
$ cat basisname
echo ${1/#*/}
$ chmod u+x basisname
$ basisname $HOME
egon
$ basisname /usr/bin/vi
vi
$
```

---

1. Siehe Dateinamen-Expandierung in Kapitel 5.9.



Um den Basisnamen eines aufgerufenen Skripts zu ermitteln, müsste im Skript folgendes angegeben werden:

```
${0##*/}
```

Mit dem Eintrag der beiden nachfolgenden Zeilen in *.profile* können für verschiedene Layers (bei Verwendung von **shl**) unterschiedliche History-Dateien eingerichtet werden:

```
vterminal=$(tty)
HISTFILE=$HOME/${vterminal##*/}
```

### Entfernen eines kleinen rechten Teilstrings

```
${variable%pattern}
```

Wenn der über *pattern*<sup>1</sup> angegebene reguläre Ausdruck das Ende des in *variable* gespeicherten Strings abdeckt, liefert dieser Ausdruck den Wert von *variable* ohne den **kleinstmöglichen** durch *pattern* abgedeckten rechten Teilstring, andernfalls liefert dieser Ausdruck den vollständigen Wert von *variable*.

#### Beispiel

```
$ text=Wasserfrosch
$ echo ${text%s*}
Wasserfro
$ echo ${text%*s}
Wasserfrosch
$ echo ${text%Wasser}
Wasserfrosch
$ cat diriname
echo ${1%/*}
$ chmod u+x diriname
$ diriname $HOME
/user1
$ echo $PWD
/user1/egon/kshueb
$ diriname $PWD
/user1/egon
$ diriname /usr/bin/vi
/usr/bin
$
```

---

1. Siehe Dateinamen-Expandierung in Kapitel 5.9.

Der nachfolgende Ausschnitt aus einem Shell-Skript zeigt eine praktische Anwendung dieser Konstruktion. Es wird dabei geprüft, ob der *dateiname* mit *.c* endet. Wenn ja, dann wird diese Datei kompiliert:

```
.....
if [${dateiname%.c} != $dateiname]
then # dateiname endet auf .c
 cc -c $dateiname
fi
.....
```

### Entfernen eines großen rechten Teilstrings

**`${variable%%pattern}`**

Wenn der über *pattern*<sup>1</sup> angegebene reguläre Ausdruck das Ende des in *variable* gespeicherten Strings abdeckt, so liefert dieser Ausdruck den Wert von *variable* ohne den **größtmöglichen** durch *pattern* abgedeckten rechten Teilstring, andernfalls liefert dieser Ausdruck den vollständigen Wert von *variable*.

#### Beispiel

```
$ text=Wasserrfrosch 
$ echo ${text%%s*} 
Wa
$ echo ${text%%+([!aeoui])}  2
Wasserfro
$
```

## 5.9 Expandierung von Dateinamen auf der Kommandozeile

Wie in der Bourne-Shell gilt auch in der Korn-Shell: Beim Aufruf eines Kommandos oder Shell-Skripts wird jedes Wort der Kommandozeile von der Korn-Shell daraufhin untersucht, ob eines der Zeichen *\**, *?* oder *[* darin vorkommt. Wird ein solches Wort gefunden, so betrachtet die Shell dieses als ein sogenanntes *Pattern*, das eine Vielzahl von Dateinamen abdecken kann.

Jedes in der Kommandozeile gefundene Pattern<sup>3</sup> wird dann von der **ksh** expandiert, d.h. durch alle Dateinamen ersetzt<sup>4</sup>, die es abdeckt. Falls kein Dateiname gefunden werden kann, den ein vorgegebenes pattern abdeckt, wird das entsprechende pattern nicht expandiert und unverändert dem aufgerufenen Kom-

1. Siehe Dateinamen-Expandierung in Kapitel 5.9.

2. Schneidet von rechts her alle Nicht-Vokale ab. Das neue *pattern* *+([!aeoui])* wird im nächsten Kapitel ausführlich beschrieben.

3. Ein pattern kann am Anfang, in der Mitte oder am Ende eines Worts angegeben sein. Es ist sogar möglich, daß ein Wort nur aus einem pattern besteht.

4. Alphabetisch sortiert.

mando oder Shell-Skript übergeben. Zur Expandierung von Dateinamen stehen wieder folgende Metazeichen zur Verfügung:

| Metazeichen   | Bedeutung                                                                                                                                                                         |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>*</b>      | steht für "eine beliebige Zeichenfolge" (auch die leere)                                                                                                                          |
| <b>?</b>      | steht für "ein beliebiges einzelnes Zeichen"                                                                                                                                      |
| <b>[...]</b>  | steht für "eines der in [...] angegebenen Zeichen".<br>Bei der Angabe von Zeichen innerhalb von [...] sind auch Bereichsangaben wie [A-Z] oder [0-9] erlaubt.                     |
| <b>[!...]</b> | steht für "ein Zeichen, welches <b>nicht</b> in [...] angegeben ist".<br>Bei der Angabe von Zeichen innerhalb von [...] sind auch Bereichsangaben wie [!B-S] oder [!4-8] erlaubt. |

Tabelle 5.5: Zur Bourne-Shell identische Metazeichen für Dateinamen-Expandierung

In **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, können noch folgende Pattern-Konstruktionen angegeben werden:

| Pattern                         | Bedeutung                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>?(pattern[  pattern]...)</b> | deckt kein oder ein Vorkommen der angegebenen Pattern ab.<br>Beispiele:<br><b>kap?(11 1[4-6] 20).1</b><br>deckt die Strings <i>kap.1</i> , <i>kap11.1</i> , <i>kap14.1</i> , <i>kap15.1</i> , <i>kap16.1</i> und <i>kap20.1</i> ab.<br><b>text.?(ein aus ksh lst)</b><br>deckt die Strings <i>text.</i> , <i>text.ein</i> , <i>text.aus</i> , <i>text.ksh</i> und <i>text.lst</i> ab. |
| <b>*(pattern[  pattern]...)</b> | deckt kein, ein oder mehrere Vorkommen der angegebenen Pattern ab.<br>Beispiele:<br><b>kap*([0-9])</b><br>deckt den String <i>kap</i> und alle Strings ab, die mit <i>kap</i> beginnen und danach beliebig viele Ziffern enthalten.                                                                                                                                                   |

Tabelle 5.6: Neue ksh-Metazeichen für Dateinamen-Expandierung

| Pattern                               | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>+(pattern[  pattern]...)</code> | <p>deckt ein oder mehrere Vorkommen der angegebenen Pattern ab.</p> <p>Beispiele:</p> <p><code>kap+([0-9])</code></p> <p>deckt alle Strings ab, die mit <i>kap</i> beginnen und danach eine oder mehrere Ziffer enthalten.</p> <p><code>*+([0-9][0-9] [a-z][a-z])</code></p> <p>deckt alle Strings ab, die mit zwei Ziffern oder zwei Kleinbuchstaben enden.</p> <p><code>*+([!a-zA-Z_0-9])*</code></p> <p>deckt alle Strings ab, in denen ein Zeichen vorkommt, das kein Buchstabe, kein Unterstrich und keine Ziffer ist.</p>                         |
| <code>@(pattern[  pattern]...)</code> | <p>deckt genau ein Vorkommen der angegebenen Pattern ab.</p> <p>Beispiele:</p> <p><code>rm *@(.bak .sik)</code></p> <p>löscht alle Dateien, deren Name mit <i>.bak</i> oder <i>.sik</i> endet.</p> <p><code>hinter@(tuer hof)</code></p> <p>deckt die beiden Strings <i>hintertuer</i> und <i>hinterhof</i> ab.</p>                                                                                                                                                                                                                                     |
| <code>!(pattern[  pattern]...)</code> | <p>deckt die Strings ab, die durch keines der angegebenen Pattern abgedeckt werden.</p> <p>Beispiele:</p> <p><code>kap*!(.[0-9])</code></p> <p>deckt die Strings ab, die mit <i>kap</i> beginnen und nicht mit einem Punkt gefolgt von einer Ziffer enden.</p> <p><code>cp *!(.[co]) /tmp</code></p> <p>kopiert alle Dateien, deren Name nicht mit <i>.c</i> oder <i>.o</i> endet in das Directory <i>/tmp</i>.</p> <p><code>!([a-f]*)</code></p> <p>deckt alle Strings ab, die nicht mit einem der Kleinbuchstaben <i>a</i> bis <i>f</i> beginnen.</p> |

Tabelle 5.6: Neue ksh-Metazeichen für Dateinamen-Expandierung (Fortsetzung)

Ausnahmen zu obigen Regeln:

Wie in der Bourne-Shell gilt auch in der Korn-Shell, daß einige Zeichenfolgen in Dateinamen nur dann abgedeckt werden, wenn sie explizit im entsprechenden Pattern angegeben wurden:

```
. (Punkt) am Anfang eines Dateinamens
/.
/
```

**Hinweis**

Die Dateinamen-Expandierung kann auch ausgeschaltet werden, indem entweder beim **ksh**-Aufruf die Option **-f** angegeben wird oder aber diese Option mit dem builtin-Kommando **set** (**set -f** oder **set -o noglob**<sup>1</sup>) eingeschaltet wird.

## 5.10 Quoting

Wie in der Bourne-Shell steht auch in der Korn-Shell der Quoting-Mechanismus zur Verfügung, um die Sonderbedeutung von Metazeichen<sup>2</sup> auszuschalten.

### 5.10.1 Verschiedene Quoting-Arten

Auch in der Korn-Shell existieren drei verschiedene Quoting-Arten:

1. Voranstellen von \
2. Klammerung mit '..'
3. Klammerung mit '"..'"

#### Voranstellen von \

Es gelten folgende Regeln:

- ▶ Wird einem der Metazeichen ein \ vorangestellt, so verliert dieses Metazeichen seine Sonderbedeutung. Wird ein \ vor einem Neuezeile-Zeichen angegeben, so wird dieses Zeichenpaar von der **ksh** entfernt. Wenn sich also eine Kommandozeile über mehr als eine Zeile erstrecken soll, so kann \ als Fortsetzungszeichen verwendet werden; in Kommentaren oder innerhalb von '..' wird \ allerdings nicht als Fortsetzungszeichen interpretiert.
- ▶ Ein \ vor einem Nicht-Metazeichen hat keinerlei Auswirkung, sondern wird lediglich entfernt.
- ▶ In einem Kommentar und innerhalb von '..' hat \ keine Sonderbedeutung.

---

1. Siehe Kapitel 5.24.3.

2. Die Metazeichen der Korn-Shell wurden im Kapitel 5.3 vorgestellt.

- ▶ Innerhalb von `".."` schaltet `\` nur die Sonderbedeutung der Metazeichen `$`, ```, `\` und `"` aus.
- ▶ Innerhalb der alten Kommandosubstitution (``kdos``) schaltet `\` nur die Sonderbedeutung der Metazeichen `$`, ``` und `\` aus.

### Klammerung mit `'..'`

Alle Metazeichen zwischen zwei einzelnen Apostrophen<sup>1</sup> (außer ein weiterer Apostroph) verlieren ihre Sonderbedeutung.

Innerhalb von `'..'` verliert sogar das zuvor vorgestellte Quoting-Zeichen `\` seine Sonderbedeutung. Sind in einer Kommandozeile viele Metazeichen auszuschalten, dann ist diese Art des Quotings dem Quoting mit `\` vorzuziehen, da sich in diesem Fall doch wesentlich lesbarere Kommandozeilen ergeben.

Zwar kann auch mit dieser Quoting-Art die Sonderbedeutung des Neuezeile-Zeichens (Abschluß einer Kommandozeile) ausgeschaltet werden; allerdings wird anders als bei `\`  der dadurch erzeugte Zeilenvorschub nicht von der Shell entfernt.

```
$ echo $H0' 
> 'ME' 
ME
$
```

[Ausgabe von `$H0`: leere Zeichenkette]

[Ausgabe des Strings `ME` in einer neuen Zeile]

Um die Sonderbedeutung eines einfachen Apostrophs innerhalb einer mit `'..'` geklammerten Zeichenkette auszuschalten, muß dieser mit Anführungszeichen geklammert werden: `'..'"..'`. Außerhalb von `'..'` kann die Sonderbedeutung von `'` mit `\` oder mit `""` ausgeschaltet werden.

### Klammerung mit `".."`

Bei einer Klammerung mit `".."` verlieren die meisten, aber nicht alle Metazeichen ihre besondere Bedeutung: Innerhalb von `".."` behalten nur die Metazeichen

```
\ " ` $
```

ihre Sonderbedeutung.

Im Unterschied zur Apostrophen-Klammerung schaltet diese Form der Klammerung also folgendes nicht aus:

- ▶ das Quoting mit `\`
- ▶ Parametersubstitution (`$variable`: Zugriff auf den Wert von `variable`)
- ▶ alte Kommandosubstitution (``kdos``)
- ▶ neue Kommandosubstitution (`$(kdos)`)

1. Nicht zu verwechseln mit den Gegen-Apostrophen der Kommandosubstitution ``..``

Da auch `\` innerhalb von `..` seine Sonderbedeutung behält, kann es verwendet werden, um die Sonderbedeutung der vier Zeichen `\` `"` ``` `$` innerhalb von `..` auszuschalten.

**Beispiel**

```
$ echo "$PWD \" \"$PWD \\$PWD \\$PWD"
/user1/egon/kshueb " $PWD \user1/egon/kshueb \ $PWD
$
```

**Hinweis**

Oft ist es notwendig, Kommandosubstitutionen mit `..` zu klammern, um bei den dadurch bereitgestellten Zeichenketten die darin enthaltenen Metazeichen (wie z.B. Dateinamen-Expandierung) auszuschalten.

Die Erkennung von builtin-Kommandos<sup>1</sup> durch die Shell kann niemals durch Quoting unterbunden werden:

```
$ \p\w\ld
/user1/egon/kshueb
$ "pwd"
/user1/egon/kshueb
$ 'pwd'
/user1/egon/kshueb
$ "cd" /bin
$ 'pwd'
/bin
$ \c\ld
$ \c'd' kshueb
$ \p"wd"
/user1/egon/kshueb
$ "cd /bin"
ksh: cd /bin: not found2
$
```

Die Erkennung der **ksh**-Schlüsselwörter<sup>3</sup> dagegen kann mit Quoting verhindert werden, z.B. `"for"`, `\for`, `"'for'"` oder `for"`.

Die Erkennung von **alias**-Namen<sup>4</sup> kann ebenfalls mit Quoting unterbunden werden.

- 
1. In Kapitel 5.24.5 sind alle builtin-Kommandos der **ksh** zusammengefaßt.
  2. String `"cd bin"` wird als Kommandoname interpretiert (existiert nicht).
  3. Siehe Kapitel 5.16.
  4. Siehe Kapitel 5.17.

## 5.10.2 Zusammenfassung der Quoting-Regeln

Die nachfolgende Tabelle gibt eine Übersicht über die Gültigkeit bestimmter Metazeichen bei den unterschiedlichen Quoting-Arten und der Kommandosubstitution:

|         | Metazeichen |    |        |   |   |   |                                                                                     |
|---------|-------------|----|--------|---|---|---|-------------------------------------------------------------------------------------|
| Quoting | \           | \$ | *?[+!@ | ` | " | ' |  |
| \       | -           | -  | -      | - | - | - | -                                                                                   |
| ".."    | x           | x  | -      | x | + | - | v                                                                                   |
| '..'    | -           | -  | -      | - | - | + | v                                                                                   |
| `..`    | x           | x  | x      | + | x | x | x                                                                                   |

Hierbei bedeutet:

- Sonderbedeutung ausgeschaltet
- x behält seine Sonderbedeutung
- + beendet entsprechendes Quoting bzw. Kommandosubstitution
- v Sonderbedeutung (Kommandoabschluß) ausgeschaltet, aber Bedeutung »Zeilenvorschub« bleibt erhalten

## 5.11 Ein- und Ausgabeumlenkung

Die üblichen Voreinstellungen sind dabei wie in der Bourne-Shell:

Standardeingabe: Dialogstation (Tastatur)  
 Standardausgabe: Dialogstation (Bildschirm)  
 Standardfehlerausgabe: Dialogstation (Bildschirm)

Jedem dieser drei Ein- / Ausgabekanäle ist wieder ein Dateideskriptor<sup>1</sup> zugeordnet:

Standardeingabe (**stdin**): 0  
 Standardausgabe (**stdout**): 1  
 Standardfehlerausgabe (**stderr**): 2

Anderen Dateien, welche innerhalb eines Programms oder Kommandos explizit eröffnet werden, werden die Dateideskriptoren 3, 4, 5, 6, usw. zugeordnet.

1. Erinnerung: In UNIX werden Geräte wie Dateien behandelt.



Diese Ein-/Ausgabekanäle können nun auch in der **ksh** in Dateien umgelenkt werden:

### <datei

lenkt die Standardeingabe (von der Dialogstation) in die Datei *datei* um.

### >datei

### >|datei

lenkt die Standardausgabe (von der Dialogstation) in die Datei *datei* um.

Existiert *datei* noch nicht, so wird sie neu angelegt. Wenn die Datei *datei* bereits existiert, so wird unterschieden, ob die Option **noclobber**<sup>1</sup> gesetzt ist oder nicht:

- ▶ gesetzt:  
**ksh** meldet einen Fehler. Bei der Angabe >|*datei* wird allerdings auch, wenn **noclobber** gesetzt ist, der alte Inhalt von *datei* überschrieben.
- ▶ nicht gesetzt:  
Der alte Inhalt von *datei* wird überschrieben.

Die Eingabe

>*datei* oder >|*datei*

ohne Angabe eines Kommandos erzeugt eine leere Datei mit Namen *datei*. Um ein Umlenken der Standardausgabe für die momentan aktive **ksh** zu erreichen, muß das builtin-Kommando **exec** verwendet werden.

#### Beispiel

```
$ pwd
/user1/egon/kshueb
$ >neudat
$ ls n*
neudat
$ set -o noclobber
$ >neudat
ksh[2]: neodat: file already exists
$ >|neudat
$ exec >logfile
$ echo "Working-Directory: $(pwd)"
$ ls -CF a*
$ exec >/dev/tty [Versuch: Standardausgabe wieder auf Bildschirm lenken]
ksh: /dev/tty: file already exists
$ exec >|/dev/tty [Standardausgabe wieder auf Bildschirm lenken]
$ cat logfile
Working-Directory: /user1/egon/kshueb
abc abc2 add* addiere.c ausgab*
$
```

---

1. Siehe Kapitel 5.24.3; **noclobber** ist jedoch nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurde.

**>>datei**

lenkt die Standardausgabe (von der Dialogstation) in die Datei *datei* um; allerdings wird hierbei der alte Inhalt einer eventuell schon existierenden Datei *datei* nicht überschrieben. Die neuen Ausgabedaten werden an das Ende von *datei* geschrieben. Sollte die Datei *datei* noch nicht existieren, wird sie wie bei der Konstruktion *>datei* bzw. *>|datei* neu angelegt.

**<<wort**

Hier-Dokument (engl. *here document*): Es wird die Eingabe an die Korn-Shell Zeile für Zeile gelesen, bis eine Zeile gefunden wird, welche genau mit *wort* übereinstimmt, oder bis ein **EOF** gelesen wird.

Anders als in der Bourne-Shell wird hier keine Parametersubstitution, keine Kommandosubstitution und keine Dateinamen-Expandierung für *wort* durchgeführt.

Abhängig davon, ob Quoting im *wort* verwendet wird oder nicht, wird die Sonderbedeutung der Metazeichen für die Eingabezeilen ausgeschaltet oder nicht:

**Quoting im wort**

In den folgenden Eingabezeilen wird die Sonderbedeutung der Metazeichen ausgeschaltet.

**Beispiel**

```
$ cat <<""ENDE␣
> Dies ist ein Hier-Dokument␣
> $PWD ist das Working-Directory␣
> ENDE␣
Dies ist ein Hier-Dokument
$PWD ist das Working-Directory
$
```

**Kein Quoting im wort**

Für die folgenden Eingabezeilen gelten dann wie in der Bourne-Shell folgende Regeln:

1. Parameter- und Kommandosubstitution findet statt.
2. *\Neuzeile-Zeichen-Kombinationen* werden ignoriert.
3. *\* muß verwendet werden, um die Sonderbedeutung der Zeichen *\* sowie *\$* und *`* auszuschalten.

**Beispiel** `$ cat <<ENDE`  
> Dies ist ein Hier-Dokument  
> \$PWD ist das Working-Directory  
> ENDE  
Dies ist ein Hier-Dokument  
/user1/egon/kshueb ist das Working-Directory  
\$

**Hinweis** Weitere Angaben nach *wort* sind möglich:

```
$ cat <<EOF | tr 'a-z' '[A-Z]'
> Dies ist ein Hier-Dokument
> $PWD ist das Working-Directory
> EOF
DIES IST EIN HIER-DOKUMENT
/USER1/EGON/KSHUEB IST DAS WORKING-DIRECTORY
$
```

Wenn mehr als ein Hier-Dokument in einer Kommandozeile angegeben ist, dann liest **ksh** diese in umgekehrter Reihenfolge:

```
$ cat <<ENDE <<FIN
> Dies ist
> lediglich Kommentar
> ENDE
> eigentlicher
> Text
> FIN
eigentlicher
Text
$
```

### <<-*wort*

identisch zur Angabe <<*wort*, außer daß in den nachfolgenden Eingabezeilen alle führenden Tabulatorzeichen ignoriert werden.

### <& *fd*

verwendet die Datei als Standardausgabe, die mit dem Dateideskriptor *fd* (Ziffer) verbunden ist.

In **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, kann für *fd* auch der Buchstabe **p** angegeben werden; in diesem Fall wird die Standardausgabe eines Ko-Prozesses, der mit Angabe von **|&** im Hintergrund gestartet wurde, direkt an die Standardeingabe weitergeleitet.

### >& *fd*

verwendet als Standardausgabe die Datei, die mit dem Dateideskriptor *fd* verbunden ist.

In **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, kann für *fd* auch der Buchstabe **p** angegeben werden; in diesem Fall wird die Standardausgabe direkt an die Standardeingabe eines Ko-Prozesses weitergeleitet, der mit **&** im Hintergrund gestartet wurde.

Nachdem die Standardeingabe oder die Standardausgabe unter Verwendung von **exec** mit einem Filedeskriptor benannt wurde, kann ein weiterer neuer Ko-Prozeß gestartet werden.

**Beispiel**

```
$ cat wartmsg
if [$# -ne 1]
then
 echo "usage: $0 username" >&2
 exit 1
fi

until who | grep $1
do
 sleep 60
done
echo "$1 hat sich angemeldet" >&2
echo "Gib deine Meldung an $1 ein" >&2
read zeile
banner $zeile | mail $1
$ chmod u+x wartmsg
$ ksh
$ wartmsg emil [Ko-Prozeß 1 wird gestartet]
[1] 1823 [Jobnummer PID]
$ exec 5>&p [Filedeskr. 5 auf Standardeing. von Koprozeß 1
festlegen]
$ wartmsg toni [Ko-Prozeß 2 wird gestartet]
[2] 2223 [Jobnummer PID]
$ exec 6>&p [Filedeskr. 6 auf Standardeing. von Koprozeß 2
festlegen]
$
toni hat sich angemeldet
Gib deine Meldung an toni ein
$ print -u6 "Wichtig. See You soon --egon" [Zeile für Koprozeß 2]
$
emil hat sich angemeldet
Gib deine Meldung an emil ein
$ echo "Testdaten ? --egon" >&5 [Zeile für Koprozeß 1]
$
```

Die entsprechenden Meldungen werden dabei den beiden Benutzern *toni* und *emil* in **banner**-Form als mail geschickt. Im obigen Beispiel wurde gezeigt, daß die entsprechende Meldung den betreffenden Ko-Prozessen auf zwei verschiedene Arten geschickt werden kann:

```

print -ufd "meldung"1

und

echo "meldung" >&fd.

$ ksh
$ (read a b c d; print "$$: $c $b $a") |& [Start Koprozeß 1]
[1] 1334 [JobnummerPID]
$ exec 5>&p [Filedeskr. 5 auf Standardeing. von Koprozeß 1
 festlegen]
$ (read; print "$$: $REPLY") |& [Start Koprozeß 2]
[2] 1523 [JobnummerPID]
$ exec 6>&p [Filedeskr. 6 auf Standardeing. von Koprozeß 2
 festlegen]
$ date >&5 [date-Ausgabe nach Koprozeß 1]
[1] - Done (read a b c d; print "$$: $c $b $a") |&
$ print -u6 "Hallo, wie gehts" [Zeile für Koprozeß 2]
$ exec 3<&p [Filedeskr. 3 auf Standardausg. von beiden
 Koprozessen legen]
[2] - Done (read; print "$$: $REPLY") |&
$ read -ru3 zeile [Lesen einer Zeile vom Koprozeß]
$ echo $zeile
1447: 04 Jun Tue
$ read -ru3 zeile [Lesen noch einer Zeile vom Koprozeß]
$ echo $zeile
1447: Hallo, wie gehts
$

```

### <>datei

eröffnet die Datei *datei* zum Lesen von und Schreiben auf die Standardeingabe. Diese Konstruktion ist nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.

### <&-

schließt die Standardeingabe (identisch zu **</dev/null**).

### >&-

schließt die Standardausgabe.

wird oft anstelle von **>/dev/null** verwendet, wenn nur die Ausführung, aber nicht die Ausgabe eines Kommandos von Wichtigkeit ist.

Bei all diesen Notationen kann zusätzlich vor dem entsprechenden Umlenkungszeichen noch ein Dateideskriptor (Zahl) angegeben werden, der den umzulenkenden »Datenstrom« (wie z.B. 2 für Standardfehlerausgabe) festlegt:

---

1. Siehe nächstes Kapitel 5.12.

**Beispiel**

```
2>fehler
```

lenkt die Standardfehlerausgabe in die Datei *fehler* um

```
3>>fehl_sammel
```

lenkt den Filedeskriptor 3 in die Datei *fehl\_sammel* um, wobei die entsprechenden Ausgaben allerdings ans Ende der Datei *fehl\_sammel* geschrieben werden.

```
2>&-
```

schließt die Standardfehlerausgabe.

```
2>&1
```

lenkt die Standardfehlerausgabe in die Datei mit dem Dateideskriptor 1 (Standardausgabe) um.

```
0<eingab_dat
```

ist identisch zur Angabe **<eingab\_dat**

```
>/tmp/prot.$$
```

kreiert eine leere Datei *prot.pid* im Directory */tmp*.

**Hinweise**

1. Unter Verwendung von **exec** ohne Angabe von Argumenten, sondern nur mit Umlenkungsanweisungen, ist es möglich, Dateien zu öffnen und zu schließen. Als Filedeskriptoren können dabei die Ziffern 0 bis 9 verwendet werden.

**Beispiel**

```
exec 3<>/dev/tty
```

eröffnet */dev/tty* (Dialogstation) mit dem Filedeskriptor 3 zum Lesen und Schreiben:

```
$ exec 3<>/dev/tty ↵
```

```
$ echo "Hallo" >&3 ↵
```

Hallo

```
$ read -u3 zeil ↵
```

```
Das ist eine Zeile ↵
```

```
$ echo $zeil ↵
```

Das ist eine Zeile

```
$
```

```
exec 3<zaehle.txt
```

eröffnet die Datei *zaehle.txt* zum Lesen und legt 3 als Filedeskriptor für diese Datei fest:

```
$ exec 3<zaehle.txt ↵
```

```
$ read -u3 ↵
```

```
$ echo $REPLY ↵
```

/etc/magic

```
$ read -u3 zeile ↵
```

```
$ echo $zeile ↵
```

/etc/passwd

```
$
```

`exec 3<&-`  
schließt die Datei mit Filedeskriptor 3.

`exec 4<>einaus`  
eröffnet die Datei *einaus* zum Lesen und Schreiben; als Filedeskriptor für diese Datei wird 4 festgelegt.

`exec 3<&5`  
eröffnet den Filedeskriptor 3 als eine Kopie von Filedeskriptor 5.

2. Vor und nach den Umlenkungsanweisungen können beliebig viele Leer- und Tabulatorzeichen angegeben sein.
3. Die Reihenfolge, in welcher die »Umlenkungsanweisungen« angegeben sind, ist signifikant: Die Shell wertet immer »von links nach rechts aus«.
4. Die Umlenkungs-Konstruktionen werden bereits vor dem Aufruf des entsprechenden Programms von der **ksh** ausgewertet, so daß das aufgerufene Programm davon keinerlei Notiz nimmt. Nachdem die **ksh** die geforderten Umlenkungen vorgenommen hat, werden die Umlenkungsangaben nicht mehr benötigt und deshalb von der **ksh** aus der Kommandozeile entfernt. Somit werden Umlenkungsanweisungen niemals einem Programm als Argumente übergeben.
5. Umlenkungsanweisungen können an beliebiger Stelle in einem einfachen Kommando angegeben werden. Üblicherweise werden die Umlenkungen allerdings am Ende eines Kommandos angegeben.
6. Für die bei der Ein-/Ausgabeumlenkung angegebene Datei findet nur dann eine Dateinamen-Expandierung statt, wenn diese nur eine Datei liefert.

**Beispiel**

```
$ ls e*
eingabe
$ read <eingabe
$ echo $REPLY
echo "Gib was ein:"
$ ls a*
abc
abc2
add
addiere.c
ausgab
$ read <a*
ksh: a*: cannot open
$ echo "Hallo" >a*
$ ls a*
a*
abc
abc2
add
addiere.c
```

```

ausgab
$ cat a*
Hallo
$

```

7. Wenn sich ein angegebenes Kommando aus mehreren einfachen Kommandos zusammensetzt, so wertet auch die **ksh** zuerst die Umlenkungsanweisungen für das gesamte Kommando aus, bevor sie die Umlenkungsangaben für die einzelnen einfachen Kommandos auswertet. Somit ergibt sich wie in der Bourne-Shell folgende Auswertungsreihenfolge:
  - a) Auswertung für die gesamte *Kommandoliste*
  - b) Auswertung für jede einzelne *Pipeline* in der *Kommandoliste*
  - c) Auswertung für jedes einzelne Kommando in den angegebenen *Pipelines*
  - d) Auswertung für jede *Kommandoliste* in jedem Kommando
8. Eine Subshell erbt die Dateideskriptoren der aufrufenden Shell.
9. Bei der Verwendung von **>&** ist auch in der **ksh** die Reihenfolge der Angabe wichtig:

```

kdo 2>&1 1>sammel
Standardfehlerausgabe ---> Bildschirm
Standardausgabe ---> sammel

kdo 1>sammel 2>&1
Standardausgabe ---> sammel
Standardfehlerausgabe ---> sammel

```

## 5.12 Die builtin-Kommandos read und print

Das Lesen vom Terminal oder aus Dateien erfolgt mit dem builtin-Kommando **read**. Anders als in der Bourne-Shell verfügt das **read** der **ksh** über Optionen.

Das Schreiben auf das Terminal oder in Dateien ist in der Korn-Shell mit dem builtin-Kommando **print** oder mit dem Alias **echo**<sup>1</sup> möglich.

### 5.12.1 Lesen von Terminals und Dateien

Um eine Zeile von der Standardeingabe zu lesen, ist das builtin-Kommando **read** zu verwenden:

```
read [-prsu[n]] [variable(n)]
```

1. **echo** ist kein builtin-Kommando der **ksh**, sondern ein Alias (siehe Kapitel 5.17).



Das Kommando **read** liest eine Zeile<sup>1</sup> von der Standardeingabe. Das erste Wort der Eingabezeile wird dann der zuerst angegebenen *variable*, das zweite der zweiten *variable* usw. zugewiesen. Zur Aufteilung der Eingabezeile in einzelne Wörter werden die Trennzeichen aus der Shell-Variable **IFS** verwendet.

Wenn mehr Worte als *variable(n)* angegeben sind, dann werden alle restlichen Worte der zuletzt angegebenen *variable* zugewiesen.

Sind mehr *variable(n)* angegeben, als Worte in einer Eingabezeile vorhanden sind, so wird den zuviel angegebenen Variablen der Nullwert (leerer String) zugewiesen.

Wird **read** ohne Angabe von *variable(n)* aufgerufen, so wird die vollständige Eingabezeile in der Variablen **REPLY** gespeichert.

Die erste Variable kann in der Form

*variable?prompt*

angegeben werden. Wenn die entsprechende **ksh** interaktiv ist, wird in diesem Fall der *prompt*-String vor dem Einlesen der entsprechenden Eingabezeile auf die Standardfehlerausgabe ausgegeben.

**read** liefert nur dann einen von 0 verschiedenen exit-Status (nicht erfolgreich), wenn **EOF** gelesen wird.

## Optionen

| Option       | Bedeutung                                                                                                                                                                                                                                                                        |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-p</b>    | liest die Eingabezeile vom Ko-Prozeß. Ein Lesen von <b>EOF</b> bewirkt dabei, daß die Verbindung zum Ko-Prozeß abgebrochen wird; danach könnte also ein neuer Ko-Prozeß kreiert werden.                                                                                          |
| <b>-r</b>    | schaltet die Sonderbedeutung von \ als Zeilen-Fortsetzungszeichen aus.                                                                                                                                                                                                           |
| <b>-s</b>    | entsprechende Eingabezeile wird als Kommando in der History-Datei <sup>a</sup> festgehalten.                                                                                                                                                                                     |
| <b>-u[n]</b> | bewirkt ein Lesen von der Datei mit Filedeskriptor <i>n</i> ; Voreinstellung für <i>n</i> ist 0 (Standardeingabe). Bevor ein anderer Filedeskriptor als 0 oder 2 verwendet werden kann, muß mit <b>exec</b> ein entsprechender Filedeskriptor eingerichtet werden <sup>b</sup> . |

a. Siehe Kapitel 5.19.

b. Siehe Kapitel 5.11.

1. Eine Eingabezeile kann sich dabei über mehrere Zeilen erstrecken, wenn als letztes Zeichen einer Teilzeile das Zeichen \ vor dem Neuezeile-Zeichen angegeben wird.

**Hinweis**

Bei der Eingabe für ein **read**-Kommando können Metazeichen der Shell durch Voranstellen von \ ausgeschaltet werden. Der Backslash wird entfernt, bevor die entsprechenden Worte den angegebenen *variable(n)* zugewiesen werden.

Ist bei der Eingabe ein builtin-Editor eingeschaltet, so können entsprechende Editor-Direktiven<sup>1</sup> bei der Eingabe der Zeile verwendet werden.

**Beispiel**

```
read -r
```

liest eine Zeile in die Variable **REPLY**

```
read -rs
```

liest eine Zeile in die Variable **REPLY** und speichert diese Zeile zusätzlich in der History-Datei

```
read -r zeile
```

liest eine Zeile in die Variable **zeile**

```
read -r zeile?"Gib ein:"
```

liest eine Zeile in die Variable **zeile**; hierbei wird vor der Eingabe die Aufforderung *Gib ein*: ausgegeben.

```
read -u3 text
```

liest aus der Datei mit Filedeskriptor 3 eine Zeile in die Variable **text**.

```
$ cat einles␣
read nachnam?"Wie heisst du ? (Nachname Vorname) " vornam
echo "Du heisst also $vornam $nachnam"
$ chmod u+x einles␣
$ einles␣
Wie heisst du ? (Nachname Vorname) Haller Sascha␣
Du heisst also Sascha Haller
$ cat pruef␣
typeset -L1 -l antwort # Konsistenzpruefg. fuer Eingaben
while read antwort?"Bitte geben Sie j oder n ein: "
 case $antwort in
 [jn]) false;;
 *) true;;
 esac
do
 echo "Falsche Eingabe"
done
echo "Eingabe ok"
$ chmod u+x pruef␣
$ pruef␣
Bitte geben Sie j oder n ein: Yes␣
Falsche Eingabe

Bitte geben Sie j oder n ein: klar␣
```

---

1. Siehe Kapitel 5.20.

Falsche Eingabe

Bitte geben Sie j oder n ein: Jaja   
Eingabe ok  
\$

## 5.12.2 Schreiben auf Terminals und in Dateien

Zum Schreiben auf die Standardausgabe steht das builtin-Kommando **print** zur Verfügung:

```
print [-Rnprsu[n]] [argument(e)]
```

Das Kommando **print** gibt die *argument(e)* auf die Standardausgabe aus, nachdem für diese Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung durchgeführt wurde. Bei der Ausgabe werden alle Argumente durch ein Leerzeichen voneinander getrennt, und diese gesamte Ausgabe wird dann mit einem Neuezeile-Zeichen abgeschlossen.

### Spezielle Notationen

Das **print**-Kommando läßt bestimmte C-ähnliche Notationen zu:

| Notation   | Bedeutung                                                                                                                                                                                                      |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>\a</b>  | Zeichen für akustisches Terminalsinal <sup>a</sup> .                                                                                                                                                           |
| <b>\b</b>  | Backspace.                                                                                                                                                                                                     |
| <b>\c</b>  | Gibt die angegebenen <i>argument(e)</i> bis zu diesem Zeichen aus und unterdrückt den üblichen Zeilenvorschub durch das Kommando <b>print</b> . Eventuell danach angegebene Argumente werden nicht ausgegeben. |
| <b>\f</b>  | Seitenvorschub ( <i>form feed</i> ).                                                                                                                                                                           |
| <b>\n</b>  | Neuezeile-Zeichen; an den Anfang der nächsten Zeile positionieren.                                                                                                                                             |
| <b>\r</b>  | <i>Carriage-Return</i> ; an den Anfang der momentanen Zeile positionieren.                                                                                                                                     |
| <b>\t</b>  | Tabulatorzeichen.                                                                                                                                                                                              |
| <b>\v</b>  | Vertikales Tabulatorzeichen.                                                                                                                                                                                   |
| <b>\\</b>  | Backslash.                                                                                                                                                                                                     |
| <b>\0n</b> | für <i>n</i> ist eine ein-, zwei- oder drei-ziffrige Oktalzahl anzugeben; das dieser Zahl entsprechende ASCII-Zeichen wird dann ausgegeben.                                                                    |

a. Ist nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurde.

## Optionen

| Option       | Bedeutung                                                                                                                                                                                                                                                                                                                                                                          |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -            | Alle nach dem Minuszeichen - angegebenen Wörter werden als Argumente interpretiert; sogar, wenn sie mit einem - (Minus) beginnen.                                                                                                                                                                                                                                                  |
| <b>-R</b>    | Schaltet die Bedeutung der oben angegebenen speziellen Notationen aus und bewirkt, daß alle nach <b>-R</b> angegebenen Wörter (außer <b>-n</b> ) als Argumente interpretiert werden, auch wenn sie mit einem - (Minus) beginnen.                                                                                                                                                   |
| <b>-n</b>    | Keinen Zeilenvorschub nach der Ausgabe der <i>argument(e)</i> .                                                                                                                                                                                                                                                                                                                    |
| <b>-p</b>    | Leitet die Ausgabe der <i>argument(e)</i> direkt an den Ko-Prozeß weiter.                                                                                                                                                                                                                                                                                                          |
| <b>-r</b>    | Schaltet die Bedeutung der oben angegebenen speziellen Notationen aus.                                                                                                                                                                                                                                                                                                             |
| <b>-s</b>    | Schreibt die <i>argument(e)</i> in die History-Datei.                                                                                                                                                                                                                                                                                                                              |
| <b>-u[n]</b> | Schreibt die <i>argument(e)</i> in die Datei mit dem Filedeskriptor <i>n</i> . <i>n</i> muß dabei 1, 2 oder ein mit <b>exec</b> eingerichteter Filedeskriptor sein <sup>a</sup> . Die Option <b>-u</b> hat dieselbe Wirkung wie ein Umlenken der Standardausgabe von <b>print</b> ; jedoch wird bei <b>-u</b> die entsprechende Datei nicht automatisch geöffnet oder geschlossen. |
|              | Voreinstellung für <i>n</i> ist 1.                                                                                                                                                                                                                                                                                                                                                 |

a. Siehe Kapitel 5.11.

Der exit-Status des **print**-Kommandos ist immer 0 (erfolgreich).

**Beispiel**

```
$ print -r 'Hallo\\theo' ↵ [print -r wird mit\\theo aufgerufen]
Hallo\\theo
$ print 'Hallo\\theo' ↵ [print wird mit\\theo aufgerufen]
Hallo\theo
$ print 'Hallo\\theo' ↵ [print wird mit\\theo aufgerufen]
Hallo\ heo
$ print -r Hallo\\theo ↵ [print -r wird mit\\theo aufgerufen]
Hallo\\theo
$ print Hallo\\theo ↵ [print wird mit\\theo aufgerufen]
Hallo\theo
$ print Hallo\\theo ↵ [print wird mit\\theo aufgerufen]
Hallo heo
$
```

**Hinweis** Auch das in der Bourne-Shell verfügbare Kommando **echo** kann in der Korn-Shell verwendet werden. Allerdings ist **echo** in der Korn-Shell kein builtin-Kommando, sondern ein Alias, so daß gilt:

**echo** ist äquivalent zu **print -**

## 5.13 Überprüfen von Bedingungen

Zum Überprüfen von Bedingungen bietet auch die **ksh** das builtin-Kommando **test** an, das bis auf einige Erweiterungen identisch zum **test**-Kommando der Bourne-Shell ist.

Allerdings bieten **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, noch ein weiteres Kommando **[.]** zum Prüfen von Bedingungen an, das das builtin-Kommando **test** überflüssig macht.

### 5.13.1 Das builtin-Kommando test

Die Aufrufsyntax für das builtin-Kommando **test** ist:

**test** *ausdr*

oder die alternative Angabe

**[** *ausdr* **]**<sup>1</sup>

Der angegebene Ausdruck *ausdr* wird dabei ausgewertet. Ist die über *ausdr* angegebene Bedingung erfüllt, so liefert **test** bzw. **[.]** den exit-Status 0 (wahr oder erfolgreich), ansonsten einen von 0 verschiedenen Wert (falsch oder nicht erfolgreich). Wenn keine Argumente angegeben sind, dann liefert **test** einen exit-Status verschieden von 0 (erfolgreich).

Es können wieder die folgenden Ausdrücke für *ausdr* angegeben werden:

| Ausdruck               | liefert wahr (exit-Status 0), wenn                                                         |
|------------------------|--------------------------------------------------------------------------------------------|
| <b>-r</b> <i>datei</i> | <i>datei</i> existiert und gelesen ( <b>read</b> ) werden darf.                            |
| <b>-w</b> <i>datei</i> | <i>datei</i> existiert und beschrieben ( <b>write</b> ) werden darf.                       |
| <b>-x</b> <i>datei</i> | <i>datei</i> existiert und ausführbar ( <b>execute</b> ) ist.                              |
| <b>-f</b> <i>datei</i> | <i>datei</i> existiert und eine normale Datei ( <b>file</b> ) ist.                         |
| <b>-d</b> <i>datei</i> | <i>datei</i> existiert und ein Directory ( <b>directory</b> ) ist.                         |
| <b>-c</b> <i>datei</i> | <i>datei</i> existiert und eine "zeichenspezifische Gerätedatei" ( <b>character</b> ) ist. |
| <b>-b</b> <i>datei</i> | <i>datei</i> existiert und eine "blockspezif. Gerätedatei" ( <b>block</b> ) ist.           |
| <b>-p</b> <i>datei</i> | <i>datei</i> existiert und eine named <b>pipe</b> ist.                                     |
| <b>-u</b> <i>datei</i> | <i>datei</i> existiert und set- <b>user</b> -id Bit gesetzt ist.                           |
| <b>-g</b> <i>datei</i> | <i>datei</i> existiert und set- <b>group</b> -id Bit gesetzt ist.                          |
| <b>-k</b> <i>datei</i> | <i>datei</i> existiert und sticky-Bit gesetzt ist.                                         |
| <b>-s</b> <i>datei</i> | <i>datei</i> existiert und nicht leer ( <b>space</b> ) ist.                                |

1. Nach **[** und vor **]** muß mindestens ein Leer- oder Tabulatorzeichen angegeben sein.

| Ausdruck                       | liefert wahr ( <i>exit-Status 0</i> ), wenn                                                                                                                                                    |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-t</b> [ <i>fd</i> ]        | die geöffnete Datei, deren Filedeskriptor <i>fd</i> ist, der Dialogstation ( <b>t</b> erminal) zugeordnet ist. Ist <i>fd</i> nicht angegeben, so wird hierfür der Filedeskriptor 1 angenommen. |
| <b>-z</b> <i>zkt</i>           | Länge der Zeichenkette <i>zkt</i> gleich 0 ( <b>z</b> ero) ist.                                                                                                                                |
| <b>-n</b> <i>zkt</i>           | Länge der Zeichenkette <i>zkt</i> nicht gleich 0 ( <b>n</b> ot <b>z</b> ero) ist.                                                                                                              |
| <i>zkt1</i> = <i>zkt2</i>      | Zeichenketten <i>zkt1</i> und <i>zkt2</i> identisch sind.                                                                                                                                      |
| <i>zkt1</i> != <i>zkt2</i>     | Zeichenketten <i>zkt1</i> und <i>zkt2</i> verschieden sind.                                                                                                                                    |
| <i>zkt</i>                     | <i>zkt</i> nicht die leere Zeichenkette ist.                                                                                                                                                   |
| <i>n1</i> <b>-eq</b> <i>n2</i> | die ganzen Zahlen <i>n1</i> und <i>n2</i> gleich ( <b>e</b> qual) sind.                                                                                                                        |
| <i>n1</i> <b>-ne</b> <i>n2</i> | die ganzen Zahlen <i>n1</i> und <i>n2</i> nicht gleich ( <b>n</b> ot <b>e</b> qual) sind.                                                                                                      |
| <i>n1</i> <b>-gt</b> <i>n2</i> | ganze Zahl <i>n1</i> größer als ( <b>g</b> reater <b>t</b> han) die ganze Zahl <i>n2</i> ist.                                                                                                  |
| <i>n1</i> <b>-ge</b> <i>n2</i> | ganze Zahl <i>n1</i> größer oder gleich ( <b>g</b> reater <b>e</b> qual) der ganzen Zahl <i>n2</i> ist.                                                                                        |
| <i>n1</i> <b>-lt</b> <i>n2</i> | ganze Zahl <i>n1</i> kleiner als ( <b>l</b> ess <b>t</b> han) die ganze Zahl <i>n2</i> ist.                                                                                                    |
| <i>n1</i> <b>-le</b> <i>n2</i> | ganze Zahl <i>n1</i> kleiner oder gleich ( <b>l</b> ess <b>e</b> qual) der ganzen Zahl <i>n2</i> ist.                                                                                          |

Zusätzlich können in der **ksh** noch folgende Ausdrücke angegeben werden:

| Ausdruck                           | liefert wahr ( <i>exit-Status 0</i> ), wenn                                                             |
|------------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>-L</b> <i>datei</i>             | <i>datei</i> existiert und ein symbolischer Link ( <b>L</b> ink) ist.                                   |
| <b>-O</b> <i>datei</i>             | <i>datei</i> existiert und der <i>datei</i> -Eigentümer der effektiven UID entspricht ( <b>O</b> wner). |
| <b>-G</b> <i>datei</i>             | <i>datei</i> existiert und die <i>datei</i> -Gruppe der effektiven GID entspricht ( <b>G</b> roup).     |
| <b>-S</b> <i>datei</i>             | <i>datei</i> existiert und eine spezielle Datei vom <i>socket</i> -Typ ist ( <b>S</b> ocket).           |
| <b>-o</b> <i>option</i>            | angegebene <i>option</i> eingeschaltet ist ( <b>o</b> ption).                                           |
| <i>dat1</i> <b>-nt</b> <i>dat2</i> | Datei <i>dat1</i> neuer als Datei <i>dat2</i> ist ( <b>n</b> ewer <b>t</b> han).                        |
| <i>dat1</i> <b>-ot</b> <i>dat2</i> | Datei <i>dat1</i> älter als Datei <i>dat2</i> ist ( <b>o</b> lder <b>t</b> han).                        |
| <i>dat1</i> <b>-ef</b> <i>dat2</i> | Datei <i>dat1</i> nur ein anderer Name für die Datei <i>dat2</i> ist ( <b>e</b> qual <b>f</b> ile).     |

Alle diese Ausdrücke können auch in der Korn-Shell mit den nachfolgenden **Operatoren** zu neuen Ausdrücken verknüpft werden:

| Ausdruck                      | Bedeutung                                                                                                                                                                     |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>! ausdr</code>          | Negationsoperator                                                                                                                                                             |
| <code>ausdr1 -a ausdr2</code> | AND-Operator                                                                                                                                                                  |
| <code>ausdr1 -o ausdr2</code> | OR-Operator                                                                                                                                                                   |
| <code>( ausdr )</code>        | Klammerung eines Ausdrucks:<br>Klammern werden verwendet, um andere als die voreingestellten Prioritäten (für die Operatoren) bei der Auswertung eines Ausdrucks festzulegen. |

Die voreingestellte **Prioritätsreihenfolge** (höchste zuerst) ist dabei:

( )  
!  
-a  
-o

**Hinweis**

- ▶ Die Ausdrücke **-o option**, **-O datei**, **-G datei** und **-S datei** sind nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.
- ▶ Auf **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, wird der Filedeskriptor *n* geprüft, wenn in den obigen Ausdrücken für *datei* der Pfadname **/dev/fd/n** angegeben ist.

Ansonsten gelten alle Regeln des **test**-Kommandos aus der Bourne-Shell.

### 5.13.2 Die Programmiersprach-Konstruktion **[[..]]**

Die Syntax für dieses neu hinzugekommene **ksh**-Kommando<sup>1</sup> ist:

**[[ ausdr ]]**

Der angegebene Ausdruck *ausdr* wird dabei ausgewertet. Ist die über *ausdr* angegebene Bedingung erfüllt, so liefert **[[..]]** (wie **test**) den exit-Status 0 (wahr oder erfolgreich), ansonsten einen von 0 verschiedenen Wert (falsch oder nicht erfolgreich).

Für *ausdr* können dabei alle beim **test**-Kommando vorgestellten Ausdrücke angegeben werden, wobei die folgenden Unterschiede gelten:

- ▶ Bei den Verknüpfungen von Ausdrücken ist
  - anstelle von **-a** der Operator **&&** und
  - anstelle von **-o** der Operator **||**
 zu verwenden.
- ▶ Bei **-t fd** kann der Filedeskriptor *fd* nicht weggelassen werden.

1. Ist nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.

- Anstelle der beiden **test**-Ausdrücke

`zkt1 = zkt2` und

`zkt1 != zkt2`

- können folgende Ausdrucksformen angegeben werden:

`zkt1 = pattern`    wahr, wenn *pattern*<sup>1</sup> den String *zkt1* abdeckt

`zkt1 != pattern`    wahr, wenn *pattern* nicht den String *zkt1* abdeckt

`zkt1 < zkt2`        wahr, wenn *zkt1* < *zkt2* (nach ASCII-Code) ist

`zkt1 > zkt2`        wahr, wenn *zkt1* > *zkt2* (nach ASCII-Code) ist

**Hinweis**

Auch beim Kommando **[[..]]** müssen – wie beim builtin-Kommando **test** – die einzelnen Operanden und Operatoren mit Leer- oder Tabulatorzeichen voneinander getrennt werden.

Da für den in **[[..]]** angegebenen Ausdruck keine Zerteilung in einzelne Wörter und auch keine Dateinamen-Expandierung von der **ksh** durchgeführt wird, können bei **[[..]]** viele ärgerliche Fehler, die bei der Verwendung von **test** auftreten, vermieden werden.

Zudem bestimmt die **ksh** die in **[[..]]** angegebenen Operatoren, bevor sie Parameter- und Kommandosubstitution für die einzelnen Wörter durchführt, so daß Wörter, die zu einem String substituiert werden, der mit einem Minuszeichen – beginnt, nicht mehr falsch als Operatoren interpretiert werden.

Auch müssen innerhalb von **[[..]]** die Klammern nicht mehr wie bei **test** mit Quoting ausgeschaltet werden.

Auf **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, ist somit das builtin-Kommando **test** bzw. **[..]** überflüssig geworden, da seine Funktionalität vollständig durch das neue Kommando **[[..]]** abgedeckt ist.

**Beispiel**

Alle drei folgenden Aufrufe prüfen, ob **\$1** genau den String "abc" enthält:

```
test "$1" = "Xabc"]
["$1" = "Xabc"]
[[$1 = abc]]
```

Die nachfolgenden Aufrufe prüfen, ob **\$1** den Namen des Working-Directories enthält; dieser Name kann dabei in Form eines absoluten oder relativen Pfadnamens vorliegen:

```
test "$1" -ef .
["$1" -ef .]
[[$1 -ef .]]
```

---

1. Alle in Kapitel 5.9 beschriebenen *Pattern*-Angaben sind erlaubt.



Die nachfolgenden Aufrufe prüfen, ob die Datei *datei* weder beschreibbar noch ausführbar ist:

```
test ! \(-w datei -o -x datei \)
[[! (-w datei || -x datei)]]
```

## 5.14 Arithmetische Auswertungen

Anders als die Bourne-Shell verfügt die Korn-Shell über eine eigene Arithmetik, so daß keine zeitwendige Aufrufe des nicht builtin-Kommandos **expr** mehr notwendig sind, um mathematische Berechnungen durchführen zu lassen.

**ksh** verwendet am jeweiligen System bei allen Berechnungen die größtmögliche interne Darstellungsform für ganze Zahlen (z.B. 2, 4 oder 8 Bytes). **ksh** prüft dabei niemals auf einen eventuellen *Overflow*.

### 5.14.1 Konstanten

Eine Konstante hat die Form

*[basis#]zahl*

*basis* muß eine Zahl zwischen 2 und 36 sein, und legt die Basis des Zahlensystems fest; Voreinstellung ist die Basis 10.

*zahl* muß eine nicht-negative ganze Zahl sein. Bei einer Basis, die größer als 10 ist, sind Groß- oder Kleinbuchstaben zu verwenden, um eine Ziffer größer als 9 darzustellen, z.B. **16#e** oder **16#E** stellt 14 im Hexadezimalsystem dar. Ein eventuell vorhandener Nachkommateil (Teil hinter einem Dezimalpunkt) wird abgeschnitten.

### 5.14.2 Variablen

Wenn eine Variable in einem arithmetischen Ausdruck das Integer-Attribut besitzt<sup>1</sup>, dann verwendet die **ksh** den Wert dieser Variablen. Ansonsten nimmt die **ksh** an, daß der Wert ein arithmetischer Ausdruck ist, und versucht, diesen auszuwerten. Zum Beispiel, wenn die Variable **a** den Wert **b+1**, die Variable **b** den Wert **c+2** und die Variable **c** den Wert **5** besitzt, dann liefert die Auswertung des Ausdrucks **2\*a** den Wert **16**. **ksh** kann Variablen bis zu 9 Ebenen tief auswerten.

Für eine Variable, die als Wert den Null-String besitzt, liefert eine Auswertung den Wert 0.

---

1. Mit **typeset -i** oder **integer** zugeteilt.

**Beispiel**

```
$ typeset -i c=5
$ b=c+2
$ a=b+1
$ print $a
b+1
$ typeset -i x=2*a
$ print $x
16
$ integer y
$ y=zimmer
ksh: zimmer: bad number
$ y=20*7
$ print $y
140
$
```

### 5.14.3 Arithmetischer Ausdruck

Wie in C wird der Wert eines arithmetischen Ausdrucks als TRUE gewertet, wenn er verschieden von 0, und als FALSE, wenn er 0 ist.

Ein Ausdruck kann zunächst einmal nur eine Konstante oder eine Variable sein. Unter Verwendung von Operatoren können dann aber mit Konstanten und Variablen mathematische Ausdrücke gebildet werden, die von der **ksh** zu berechnen sind. Das folgende Syntaxdiagramm soll dies nochmals verdeutlichen:

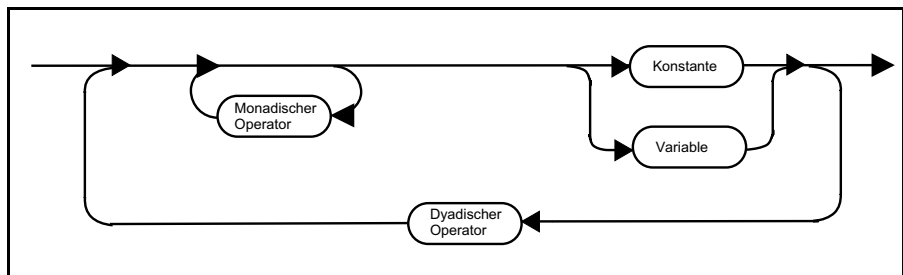


Abbildung 5.2: Syntaxdiagramm für einen Ausdruck in der ksh

Die in der **ksh** möglichen Operatoren entsprechen weitgehend den Operatoren der Programmiersprache C:

#### Monadische Operatoren

| Operator            | Bedeutung                                                                      |
|---------------------|--------------------------------------------------------------------------------|
| <code>-ausdr</code> | Minuszeichen: negiert den arithmetischen Wert von <i>ausdr</i> .               |
| <code>!ausdr</code> | Logische Negation: negiert den Wahrheitswert von <i>ausdr</i> .                |
| <code>~ausdr</code> | Bitweise Negation: bildet das Einer-Komplement zum <i>ausdr</i> <sup>a</sup> . |

a. Nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.

## Dyadische Operatoren

| Operator                 | Bedeutung                                                                                      |
|--------------------------|------------------------------------------------------------------------------------------------|
| <i>ausdr1*ausdr2</i>     | Multiplikation                                                                                 |
| <i>ausdr1/ausdr2</i>     | Division                                                                                       |
| <i>ausdr1%ausdr2</i>     | Modulo: liefert den Rest aus der ganzzahligen Division von <i>ausdr1</i> durch <i>ausdr2</i> . |
| <i>ausdr1+ausdr2</i>     | Addition                                                                                       |
| <i>ausdr1-ausdr2</i>     | Subtraktion                                                                                    |
| <i>ausdr1&lt;=ausdr2</i> | kleiner gleich                                                                                 |
| <i>ausdr1&gt;=ausdr2</i> | größer gleich                                                                                  |
| <i>ausdr1&lt;ausdr2</i>  | kleiner                                                                                        |
| <i>ausdr1&gt;ausdr2</i>  | größer                                                                                         |
| <i>ausdr1==ausdr2</i>    | gleich                                                                                         |
| <i>ausdr1!=ausdr2</i>    | ungleich                                                                                       |
| <i>var=ausdr</i>         | Zuweisung von <i>ausdr</i> an Variable <i>var</i> .                                            |

Ausdrücke mit den Vergleichsoperatoren `<=`, `>=`, `<`, `>`, `==`, und `!=` liefern TRUE (1), wenn die angegebene Bedingung erfüllt ist, ansonsten FALSE (0).

Die folgenden Operatoren sind nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.

| Operator                      | Bedeutung                                                                                                                  |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <i>ausdr1&lt;&lt;ausdr2</i>   | Links-Shift: schiebt den Wert von <i>ausdr1</i> um <i>ausdr2</i> -Bits nach links.                                         |
| <i>ausdr1&gt;&gt;ausdr2</i>   | Rechts-Shift: schiebt den Wert von <i>ausdr1</i> um <i>ausdr2</i> -Bits nach rechts.                                       |
| <i>ausdr1&amp;ausdr2</i>      | bitweises AND                                                                                                              |
| <i>ausdr1^ausdr2</i>          | bitweises XOR                                                                                                              |
| <i>ausdr1 ausdr2</i>          | bitweises OR                                                                                                               |
| <i>ausdr1&amp;&amp;ausdr2</i> | logisches AND: wenn <i>ausdr1</i> FALSE (0) liefert, so wird <i>ausdr2</i> überhaupt nicht ausgewertet.                    |
| <i>ausdr1  ausdr2</i>         | logisches OR: wenn <i>ausdr1</i> TRUE (Wert verschieden von 0) liefert, so wird <i>ausdr2</i> überhaupt nicht ausgewertet. |
| <i>var*=ausdr</i>             | entspricht: <i>var=var*ausdr</i>                                                                                           |
| <i>var/=ausdr</i>             | entspricht: <i>var=var/ausdr</i>                                                                                           |

| Operator                       | Bedeutung                                     |
|--------------------------------|-----------------------------------------------|
| <code>var%=ausdr</code>        | entspricht: <code>var=var%ausdr</code>        |
| <code>var+=ausdr</code>        | entspricht: <code>var=var+ausdr</code>        |
| <code>var-=ausdr</code>        | entspricht: <code>var=var-ausdr</code>        |
| <code>var&lt;&lt;=ausdr</code> | entspricht: <code>var=var&lt;&lt;ausdr</code> |
| <code>var&gt;&gt;=ausdr</code> | entspricht: <code>var=var&gt;&gt;ausdr</code> |
| <code>var&amp;=ausdr</code>    | entspricht: <code>var=var&amp;ausdr</code>    |
| <code>var^=ausdr</code>        | entspricht: <code>var=var^ausdr</code>        |
| <code>var =ausdr</code>        | entspricht: <code>var=var ausdr</code>        |

Die logischen Operatoren **&&** und **|** liefern TRUE (1), wenn die angegebene Gesamtbedingung erfüllt ist, ansonsten FALSE (0).

Die folgende Tabelle zeigt die voreingestellte **Prioritätsreihenfolge** (höchste zuerst). Alle in einer Zeile angegebenen Operatoren besitzen dabei die gleiche Priorität:

```
()
- (Minuszeichen)
! ~
* / %
+ -
<< >>
<= >= < >
== !=
&
^
|
&&
||
= *= /= %= += -= <<= >>= &= ^= |=
```

Außer die Zuweisungsoperatoren (in der letzten Tabellenzeile) werden alle Operatoren mit gleicher Priorität von links nach rechts abgearbeitet.

Mit Klammerung ist es möglich, eine andere als die durch die Prioritäten vorgegebene Auswertung zu erzwingen.

**Beispiel**

```
$ integer a b c
$ a=4*3/2 [entspricht: a=(4*3)/2]
$ print $a
6
$ a=4*(3/2) [(3/2) ergibt 1, da Nachkommateil abgeschnitten wird]
$ print $a
4
$ a=4*3+2 [entspricht: a=(4*3)+2]
$ print $a
```

```
14
$ a=4*(3+2)
$ print $a
20
$ b=7
$ c=b+a [entspricht: c=(b=b+a)]
$ print $a $b $c
20 27 27
$
```

**Hinweis**

Operatoren, welche Metazeichen in der **ksh** sind, müssen mit Quoting ausgeschaltet werden.

Arithmetische Ausdrücke können verwendet werden:

- ▶ als Array-Index; z.B.  
`x[4*(i+1)]=7`
- ▶ als Argument beim builtin-Kommando **shift**; z.B.  
`shift $#-2`
- ▶ als Operanden bei den arithmetischen Vergleichsoperatoren von **test**, **[..]** und **[[..]]**; z.B.  

```
$ integer i=0
$ while [[i=$i+1 -le 5]]
> do
> print $i
> done
```

1  
2  
3  
4  
5  
\$
- ▶ als Limit-Angabe beim builtin-Kommando **ulimit**<sup>1</sup>; z.B.  
`ulimit -t 60*60*2 # Zeitlimit für alle Prozesse auf 2 Stunden festlegen2`
- ▶ auf der rechten Seite einer Zuweisung an eine Integer-Variable; z.B.  

```
$ typeset -i d
$ d=7+(2*3*4*5)
$ print $d
```

127  
\$
- ▶ für jedes Argument beim Aufruf des builtin-Kommandos **let** oder innerhalb von **((..))**<sup>3</sup>.

1. Siehe Kapitel 5.24.2.

2. Option **-t** bei **ulimit** ist nicht auf allen **ksh**-Versionen verfügbar.

3. Siehe nächstes Kapitel 5.14.4.

### 5.14.4 **let** und **((..))**

Mit dem Vorhandensein des builtin-Kommandos **let** ist das nicht builtin- (und damit langsamere) Kommando **expr** in der Korn-Shell nicht mehr notwendig.

Die Aufrufsyntax für **let** ist:

**let** *argument(e)*

Jedes *argument* ist dabei ein auszuwertender Ausdruck; deshalb müssen Leer- und Tabulatorzeichen in einem arithmetischen Ausdruck mit Quoting ausgeschaltet werden. Operatoren, welche Metazeichen für die **ksh** sind, sind ebenfalls mit Quoting auszuschalten.

Der Rückgabewert von **let** ist 0 (erfolgreich), wenn die Auswertung des letzten Ausdrucks einen Wert verschieden von 0 liefert. Andernfalls ist der Rückgabewert von **let** 1 (nicht erfolgreich).

Ist nur ein Ausdruck auszuwerten, so kann anstelle von **let** das Kommando

**((ausdr))**

verwendet werden, welches dem Aufruf

**let "ausdr"**

entspricht, so daß bei Verwendung von Leer-, Tabulator- oder sonstigen Metazeichen in *ausdr* diese nicht mit Quoting auszuschalten sind.

**((..))** wird meist bei der Bedingungsangabe der Kommandos **if**, **while** und **until** benutzt.

Eine andere Verwendung von **((..))** ist das Rechnen mit Variablen, die zwar arithmetische Ausdrücke als Werte beinhalten, aber nicht explizit als Integer-Variablen deklariert wurden.

#### Beispiel

```
$ cat num[↵]
typeset -R5 zeilnr=1
IFS="" # Um Leerzeichen aus der Eingabe als echte
 # Zeichen und nicht als Trennzeichen
 # interpretieren zu lassen
cat $* |
while read -r zeile
do
 print -R "$zeilnr:" "$zeile"
 ((zeilnr=zeilnr+1)) # ((..))-Klammerung, da
 # zeilnr keine
 # Integer-Variable ist.
done
$ chmod u+x num[↵]
$ num addiere.c[↵]
1: #include <stdio.h>
2:
```

```
3: main()
4: {
5: float a, b;
6:
7: printf("Gib 2 Zahlen ein: ");
8: scanf("%f %f", &a, &b);
9: printf("Die Summe ist: %.3f\n", a+b);
10: }

$ cat fiba ↵
integer ende=${1:-1000} x=1 y=1 z

while ((x<=ende))
do
 print $x
 let z=x+y x=y y=z
done
$ chmod u+x fiba ↵
$ fiba ↵
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
$ fiba 10 ↵
1
1
2
3
5
8
$ cat kexpr ↵
Realisierung des Kommandos expr als Funktion
expr()
{
 integer ausdr="$*"
 print $ausdr
}
$. kexpr ↵ [Lesen der Datei kexpr mit Punkt-Kommando]
$ expr 5*8 ↵ [Hier sind keine Leerstellen zwischen den Operanden nötig]
```

```
40
$ expr 5 * 8 ↵
40
$ type expr ↵
expr is a function
$ cat vonhinten ↵
vonhinten - liest bis zu 512 Zeilen aus einer Datei
bzw. von der Standardeingabe und gibt sie
dann in umgekehrter Reihenfolge aus.
#
usage: vonhinten [dateiname]
#
IFS="" # Um Trennzeichen aus der Eingabe als
 # echte Zeichen interpretieren zu lassen

if [[$1 != ""]] # wenn eine Datei auf Kdozeile
then exec 0< $1 # angegeben ist, dann
fi # Standardeingabe dorthin umlenken
integer nr=0
Zeilen einlesen und im Array puffer speichern
while ((nr < 512)) && read -r puffer[$nr]
do ((nr=nr+1))
done

Zeilen in umgekehrter Reihenfolge ausgeben
while ((nr > 0))
do print -R ${puffer[`${nr}=1]}}
done
$ chmod u+x vonhinten ↵
$ vonhinten addiere.c ↵
}
printf("Die Summe ist: %.3f\n", a+b);
scanf("%f %f", &a, &b);
printf("Gib 2 Zahlen ein: ");

float a, b;
{
main()

#include <stdio.h>
$ vonhinten ↵
eins ↵
zwei ↵
drei ↵
Strg-D
drei
zwei
eins
$
```



## 5.15 Kommandoklammerung

Die Korn-Shell kennt die gleichen Arten von Kommandoklammerung wie die Bourne-Shell:

- ▶ runde Klammern: ( *kdoliste* ) und
- ▶ geschweifte Klammern: { *kdoliste*; }

### 5.15.1 Runde Klammern: ( *kdoliste* )

Diese Klammerung bewirkt, daß für alle in *kdoliste* angegebenen Kommandos eine Subshell gestartet wird.

Angaben wie **&** oder Umlenkungsanweisungen beziehen sich auch in der **ksh** immer nur auf das jeweilige Teilkommando. Mit der Verwendung von Klammern können nun durch den Benutzer andere als die voreingestellten Prioritäten festgelegt werden.

Der exit-Status der ganzen in ( .. ) angegebenen *kdoliste* ist der exit-Status des letzten Kommandos dieser *kdoliste*.

Auch in der **ksh** sind geschachtelte Klammerungen erlaubt. Allerdings müssen anders als in der Bourne-Shell eventuell nacheinander angegebene öffnende Klammern mit Leer-, Tabulator- oder Neuezeile-Zeichen voneinander getrennt werden, um sie vom **ksh**-Kommando ((.)), welches eine arithmetische Auswertung nach sich zieht, zu unterscheiden.

### 5.15.2 Geschweifte Klammern: { *kdoliste*; }

Im Unterschied zur (..)-Klammerung wird auch in der **ksh** die hier angegebene *kdoliste* nicht von einer neuen Subshell, sondern von der momentan aktiven Shell ausgeführt.

Wie in der Bourne-Shell sind auch in der **ksh** bei der Klammerung mit {..} zwei Syntaxregeln zu beachten:

- ▶ Das letzte Zeichen in {..} muß ein ; oder ein Neuezeile-Zeichen sein.
- ▶ Da es sich bei { um ein Schlüsselwort der **ksh** handelt, ist vor und nach { mindestens ein Trennzeichen (Leer-, Tabulator-, Neuezeile-Zeichen) anzugeben, damit die Shell es als solches erkennen kann.

Der exit-Status der ganzen in {..} angegebenen *kdoliste* ist der exit-Status des letzten Kommandos dieser *kdoliste*.

Auch hier sind wieder geschachtelte Klammerungen erlaubt.

**Beispiel**

Die nachfolgende Aufrufzeile bewirkt, daß die ersten beiden Zeilen der Datei *kexpr* in die beiden Variablen *zeile1* und *zeile2* gelesen werden und der Rest dieser Datei dann mit **cat** am Bildschirm ausgegeben wird:

```
$ { read -r zeile1; read -r zeile2; cat;} < kexpr ↵
{
 integer ausdr="$*"
 print $ausdr
}
$ print $zeile1 ↵
 # Realisierung des Kommandos expr als Funktion
$ print $zeile2 ↵
expr()
$
```

## 5.16 Programmiersprachkonstrukte der Korn-Shell

Ein Kommando in der **ksh** ist

- ▶ ein einfaches Kommando,
- ▶ eine Kommandoklammerung oder
- ▶ ein »Programmiersprachkommando«.

Die Syntaxdiagramme aus Abbildung 5.3 fassen dies nochmals zusammen.

Jedes dieser Konstrukte wird als ein einziges Kommando von der **ksh** betrachtet; somit kann auch jedes dieser Kommandos ein Element einer Pipeline oder Liste sein.

Die folgenden Wörter werden von der Shell nur dann als Schlüsselwörter erkannt, wenn sie das erste Wort eines Kommandos sind, und ihre Sonderbedeutung nicht durch Quoting ausgeschaltet ist:

```
if then else elif fi
case esac
for while until do done
{ }
select in
function
time
[[]]
```

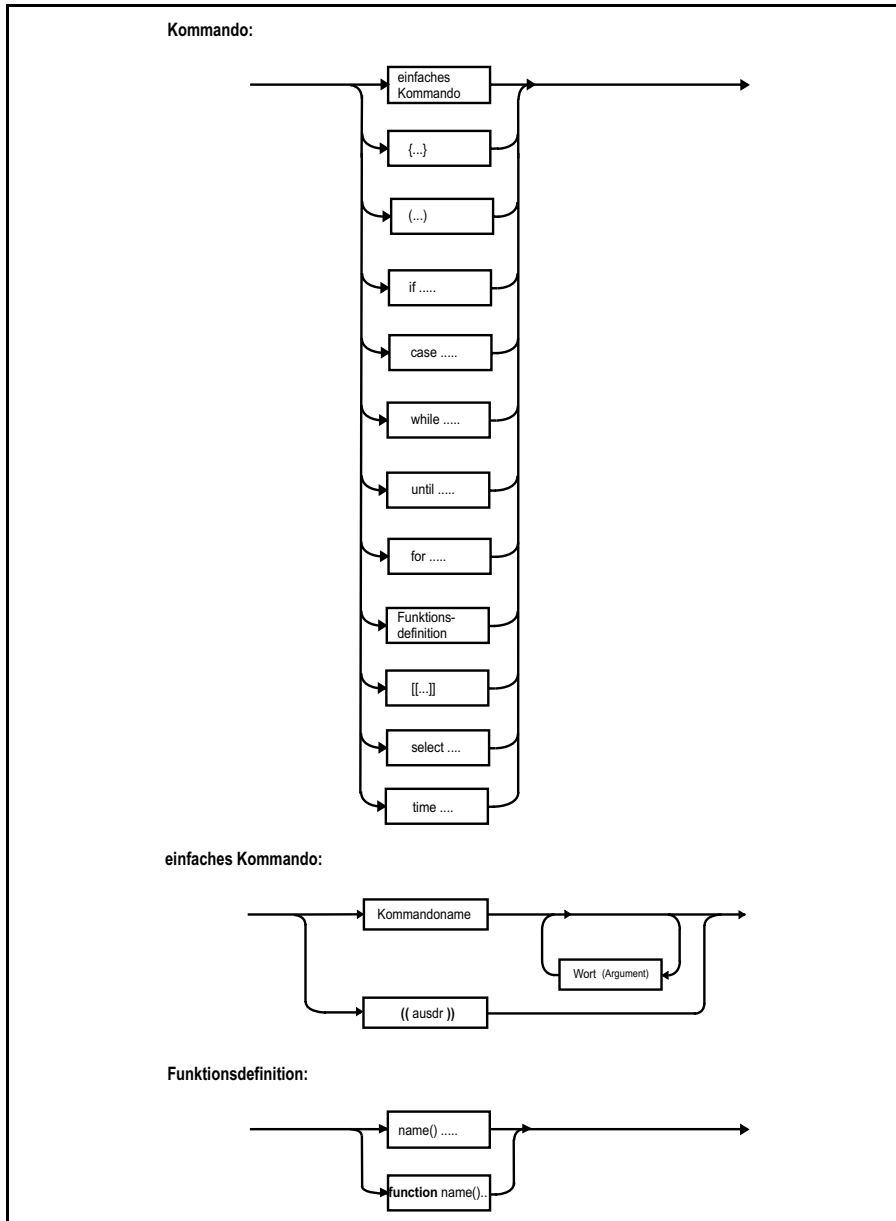


Abbildung 5.3: Syntaxdiagramme für Kommandos in der ksh

Die **ksh** erkennt diese Schlüsselwörter nur, wenn sie folgendermaßen angegeben werden:

- ▶ als erstes Wort in einer Zeile oder
- ▶ nach einem der Operatoren **;**, **|**, **||**, **&**, **&&**, **|&**, **(**, **)** oder
- ▶ als erstes Wort nach einem Schlüsselwort, außer nach **case**, **for**, **in**, **select**, **[[**

Das Schlüsselwort **in** wird auch als zweites Wort nach **case**, **for** oder **select** erkannt.

### 5.16.1 if-Anweisung

```
if if_kdolist1
then
 then_kdolist1
[elif if_kdolist2
 then
 then_kdolist2]
:
:
[else
 else_kdolist]
fi
```

entspricht vollständig dem **if**-Kommando der Bourne-Shell (siehe Kapitel 4.12.1).

#### Hinweise

1. Die einseitige **if**-Anweisung kann auch wieder unter Verwendung von **&&** und **||** nachgebildet werden.

```
if_kdolist && then_kdolist
```

entspricht

```
if if_kdolist
then
 then_kdolist
fi
```

Die Angabe

```
if_kdolist || then_kdolist
```

entspricht

```
if ! if_kdolist
then
 then_kdolist
fi
```

2. Der exit-Wert eines **if**-Kommandos ist der exit-Wert des letzten ausgeführten Kommandos (im **then**- bzw. **else**-Teil). Wenn keine Kommandos aus einem **then**- oder **else**-Teil ausgeführt wurden, dann ist der exit-Status 0 (entspricht erfolgreichem Ablauf des **if**-Kommandos).

## 5.16.2 case-Anweisung

```
case wort in
 pattern1) kdoliste1;;
 pattern2) kdoliste2;;
 :
 :
 patternn) kdolisten;;
esac
```

entspricht weitgehend dem **case**-Kommando der Bourne-Shell: Die Zeichenkette *wort* wird dabei in der angegebenen Reihenfolge zunächst mit *pattern1*, dann mit *pattern2* usw. verglichen, bis eine Übereinstimmung gefunden wird. Bei einer Übereinstimmung wird dann die zugehörige *kdoliste* ausgeführt und danach mit dem nächsten Kommando nach dem Schlüsselwort **esac** fortgefahren.

Auch hier kann wieder das Zeichen | verwendet werden, um mehrere alternative Pattern für eine *kdoliste* anzugeben.

Jedoch gibt es eine Ausnahme zu beachten: Es kann vor jeder Pattern-Angabe eine runde Klammer ( angegeben werden.

Wird in **\$(..)** eine **case**-Anweisung verwendet, so ist vor jeder Pattern-Angabe eine öffnende Klammer anzugeben, um sicherzustellen, daß gleich viele öffnende wie schließende Klammern in **\$(..)** vorhanden sind.

In **ksh**-Versionen, die vor dem 3.6.1986 freigegeben wurden, ist diese Angabe von ( vor einem **case**-pattern allerdings nicht erlaubt; das wiederum bedeutet, daß in diesen **ksh**-Versionen keine **case**-Anweisung in **\$(..)** erlaubt ist.

Zusätzlich gilt auch noch, daß in **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, für **case**-pattern auch die neu hinzugekommenen pattern<sup>1</sup> angegeben werden können:

```
?(pattern[|pattern]...)
*(pattern[|pattern]...)
+(pattern[|pattern]...)
@(pattern[|pattern]...)
!(pattern[|pattern]...)
```

---

1. Siehe Kapitel 5.9.

**Hinweis**

1. Der exit-Wert eines **case**-Kommandos ist wieder der exit-Wert der letzten ausgeführten Anweisung oder eben 0, wenn keine der angebotenen Alternativen ausgewählt wurde.
2. Die angegebenen pattern werden in der Reihenfolge der Angabe daraufhin überprüft, ob sie das vorgegebene *wort* abdecken. Deswegen sollten wieder die »default«-Angaben niemals als erste pattern angegeben werden.
3. Für *wort* wird Dateinamen-Expandierung, Kommandosubstitution und Parametersubstitution durchgeführt.

### 5.16.3 while-Schleife

```
while kdoliste1
do
 kdoliste2
done
```

entspricht vollständig dem **while**-Kommando der Bourne-Shell: Zuerst werden die Kommandos aus *kdoliste1* ausgeführt. Wenn der exit-Status dieser Kommandoliste (des letzten Kommandos) gleich 0 (erfolgreich) ist, dann werden die Kommandos aus *kdoliste2* ausgeführt. Dieser Ablauf wird wiederholt, bis die Ausführung von *kdoliste1* einen exit-Status verschieden von 0 (nicht erfolgreich) liefert. In diesem Fall wird die Abarbeitung hinter der **done**-Anweisung fortgesetzt.

Der exit-Status einer **while**-Schleife hängt wieder davon ab, ob der Schleifenkörper ausgeführt wurde oder nicht:

Wurde der Schleifenkörper ausgeführt, dann ist der exit-Status des gesamten »Schleifen-Kommandos« der exit-Status des letzten ausgeführten Kommandos im Schleifenkörper (aus *kdoliste2*).

Wurde der Schleifenkörper nicht ausgeführt, dann ist der exit-Status gleich 0 (erfolgreich).

**Hinweis**

Das builtin-Kommando **break** bewirkt, daß die **while**-Schleife mit einem exit-Status 0 (erfolgreich) verlassen wird.

Das builtin-Kommando **continue** bewirkt, daß die Ausführung der *kdoliste2* abgebrochen, und unmittelbar wieder mit der Ausführung von *kdoliste1* fortgefahren wird.

### 5.16.4 until-Schleife

```
until kdoliste1
do
 kdoliste2
done
```

entspricht vollständig dem **until**-Kommando der Bourne-Shell: Die **until**-Schleife stellt die Umkehrung zur **while**-Schleife dar: Der Schleifenrumpf (*kdoliste2*) wird solange ausgeführt wie *kdoliste1* einen exit-Wert verschieden von 0 (Ausführung des letzten Kommandos aus *kdoliste1* war nicht erfolgreich) liefert.

Der exit-Status einer **until**-Schleife hängt wieder davon ab, ob der Schleifenkörper ausgeführt wurde oder nicht:

Wurde der Schleifenkörper ausgeführt, dann ist der exit-Status des gesamten »Schleifen-Kommandos« der exit-Status des letzten ausgeführten Kommandos im Schleifenkörper (aus *kdoliste2*).

Wurde der Schleifenkörper nicht ausgeführt, dann ist der exit-Status gleich 0 (erfolgreich).

#### Beispiel

Beim Aufruf des folgenden Skripts wird für die angegebene C-Programmdatei immer wieder der Editor **vi** aufgerufen, bis dieses C-Programm fehlerfrei kompiliert wurde:

```
$ cat ccvi 
if [$# -ne 1]
then
 echo "usage: $0 c-datei"
 exit 1
fi
until cc $1
do read dummy?"Weiter mit RETURN-Taste"
 vi $1
done
$ chmod u+x ccvi 
$
```

#### Hinweise

Das builtin-Kommando **break** bewirkt, daß die **until**-Schleife mit einem exit-Status 0 (erfolgreich) verlassen wird.

Das builtin-Kommando **continue** bewirkt, daß die Ausführung von *kdoliste2* abgebrochen und unmittelbar wieder mit der Ausführung von *kdoliste1* fortgefahren wird.

## 5.16.5 for-Schleife

```
for laufvariable [in wort1 ... wortn]
do
 kdoliste
done
```

entspricht vollständig dem **for**-Kommando der Bourne-Shell: Während die Ausführung von **while**- und **until**-Schleifen an die Auswertung von Bedingungen geknüpft ist, wird bei der **for**-Schleife die Anzahl der Schleifendurchläufe durch eine bestimmte *wort*-Liste festgelegt. Als *wort*-Liste werden dabei

- ▶ entweder implizit alle Positionsparameter (keine **in**-Angabe)
- ▶ oder die danach angegebenen *worte*

genommen.

Der exit-Status einer **for**-Schleife hängt wieder davon ab, ob der Schleifenkörper ausgeführt wurde oder nicht:

Wurde der Schleifenkörper ausgeführt, dann ist der exit-Status des gesamten »Schleifen-Kommandos« der exit-Status des letzten ausgeführten Kommandos im Schleifenkörper (aus *kdoliste*).

Wurde der Schleifenkörper nicht ausgeführt, dann ist der exit-Status gleich 0 (erfolgreich).

### Hinweise

Das builtin-Kommando **break** bewirkt, daß die **for**-Schleife mit einem exit-Status 0 (erfolgreich) verlassen wird.

Das builtin-Kommando **continue** bewirkt, daß die Ausführung der *kdoliste* abgebrochen und unmittelbar mit der Ausführung des nächsten Schleifendurchlaufs fortgefahren wird.

Für die angegebenen *worte* führt die **ksh** wieder Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung durch.

## 5.16.6 Kommando [[..]]

```
[[ausdr]]
```

Dieses Kommando wurde bereits in Kapitel 5.13 genau besprochen: Es wertet den angegebenen *ausdr* aus. Ist die über *ausdr* angegebene Bedingung erfüllt, so liefert **[[..]]** (wie **test**) den exit-Status 0 (wahr oder erfolgreich), ansonsten einen von 0 verschiedenen Wert (falsch oder nicht erfolgreich).



### 5.16.7 select-Kommando

```
select variable [in wort1 ... wortn]
do
 kdoliste
done
```

Das **select**-Kommando ist neu in der **ksh**. Es wurde eigens zur Ausgabe und Bearbeitung von Menüs eingeführt:

**select** zeigt zunächst alle Auswahlmöglichkeiten eines Menüs am Bildschirm (Standardfehlerausgabe) an.

Die Auswahlmöglichkeiten sind dabei die angegebenen *worte*. Ist

**in** *wort1 ... wortn*

nicht angegeben, so legen die Positionsparameter die Auswahlmöglichkeiten fest:

**select** *variable*

entspricht somit der Angabe:

**select** *variable* **in** "\$@"

Die beiden vordefinierten Shellvariablen **COLUMNS** und **LINES**<sup>1</sup> können entsprechend gesetzt werden, um die Ausgabe des Menüs geeignet zu formatieren.

Jedem Menüpunkt wird bei der Ausgabe automatisch eine Zahl vorangestellt; z.B. würde die Angabe

```
select i in Suchen Ersetzen Sortieren Verlassen
do
 kdoliste
done
```

zu folgender Menü-Ausgabe führen:

```
1) Suchen
2) Ersetzen
3) Sortieren
4) Verlassen
#?
```

- Nach der Ausgabe des entsprechenden Menüs gibt **select** den Inhalt der Shellvariablen **PS3**<sup>2</sup> als Prompt aus, um dem Benutzer anzuzeigen, daß es für die Eingabe einer Auswahl bereit ist. **PS3** kann dabei mit einer entsprechenden Eingabeaufforderung gesetzt werden; z.B. **PS3="Wählen Sie bitte aus :"**.

Die Voreinstellung ist: **PS3="#? "**.

---

1. Siehe Kapitel 5.8.2.  
2. Siehe Kapitel 5.8.2.

- ▶ Wählt der Benutzer hierbei einen Menüpunkt über eine Zahl aus, so besetzt **select** die *variable* mit dem entsprechenden *word*, das dieser Zahl im Menü entspricht.

Gibt der Benutzer dagegen als Antwort eine leere Zeile ein, dann zeigt **select** sofort wieder das Menü und den Prompt **PS3** als Eingabeaufforderung an; in diesem Fall wird die *kdoliste* nicht ausgeführt.

Wird eine nicht erlaubte Menüauswahl getroffen, so besetzt **select** die *variable* mit dem Nullwert.

- ▶ Die wirkliche Benutzereingabe (Zahl) wird immer in der automatischen Variable **REPLY**<sup>1</sup> gespeichert.
- ▶ Das **select**-Kommando wird nur verlassen, wenn es bei der Ausführung der *kdoliste* auf ein **break**-, **return**- oder **exit**-Kommando trifft, oder es aber **EOF** (*Strg-D*) liest.
- ▶ Für die angegebenen *worte* führt die **ksh** Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung durch.

**Beispiel**

```
$ cat selecttest ↵
PS3="Deine Wahl ?"
select wahl in Suchen Ersetzen Sortieren Verlassen
do ↵
 if ["$wahl" = "Verlassen"]
 then
 break
 else
 print "Du hast -- $wahl ($REPLY) -- gewaehlt"
 fi
done
$ chmod u+x selecttest ↵
$ selecttest ↵
1) Suchen
2) Ersetzen
3) Sortieren
4) Verlassen
Deine Wahl ?2 ↵
Du hast -- Ersetzen (2) -- gewaehlt
Deine Wahl ?1 ↵
Du hast -- Suchen (1) -- gewaehlt
Deine Wahl ?0 ↵
Du hast -- (0) -- gewaehlt
Deine Wahl ? ↵
1) Suchen
2) Ersetzen
3) Sortieren
4) Verlassen
```

---

1. Siehe Kapitel 5.8.2.

```

Deine Wahl ?4
$ cat seltext
PS3="Wie soll der Text ausgegeben werden ? "
print "Gib deinen Text ein" >&2
read - text
select i in Normal Unterstrichen Spruchband \
 "Neuen Text" Ende
do
case $i in
 Normal) print - "$text";;
 Unterstrichen) print - "$text"
 integer z=1 laenge=${#text}
 while ((z<=laenge))
 do print -n - "-"
 ((z=z+1))
 done
 print;;
 Spruchband) banner "$text";;
 "Neuen Text") print "Gib deinen neuen Text ein" >&2
 read - text;;
 Ende) break;;
 *) print -n "\n\007unerlaubte Eingabe: "
 print "$REPLY\n" >&2;;

esac
done
$ chmod u+x seltext
$ seltext
Gib deinen Text ein
Der Wal
1) Normal
2) Unterstrichen
3) Spruchband
4) Neuen Text
5) Ende
Wie soll der Text ausgegeben werden ? 1
Der Wal
Wie soll der Text ausgegeben werden ? 2
Der Wal

Wie soll der Text ausgegeben werden ? 3
#
#
#
#
#
#
#####
Wie soll der Text ausgegeben werden ? 4
Gib deinen neuen Text ein
Niklas

```

```
Wie soll der Text ausgegeben werden ?
1) Normal
2) Unterstrichen
3) Spruchband
4) Neuen Text
5) Ende
Wie soll der Text ausgegeben werden ? 2
Niklas

Wie soll der Text ausgegeben werden ? 6

unerlaubte Eingabe: 6

Wie soll der Text ausgegeben werden ? 5
$ cat datanal
PS3="Ihre Wahl ? "
if [[$# -eq 0]]
then
 print "usage: $0 datei(en)" >&2
 exit 1
else
 print -n "\nInhaltsanalyse fuer folgende "
 print - "Dateien:\n$\n\n"
fi

select i in "Wortstatistik" \
 "Durchschnittliche Zeilenlaenge" \
 "Zeilen-, Wort- und Zeichenanzahl" \
 "Verlassen"
do
 case $REPLY in
 1) cat $* |
 tr -cs "[a-z][A-Z][0-9]" "[\012*]" |
 tr "[A-Z]" "[a-z]" |
 sort |
 uniq -c;;
 2) integer nr=0 ges_laenge=0 durch
 ALT_IFS="$IFS"
 cat "$@" |
 { IFS=; while read -r zeile
 do ((nr=nr+1))
 ((ges_laenge+=${#zeile}))
 done;}
 ((durch=ges_laenge/nr))
 print -n "Durchschnittliche Zeilenlaenge: "
 print $durch
 IFS="$ALT_IFS";;
 3) wc $*;;
 4) break;;
 *) print - "\n\007unerlaubte Eingabe: $REPLY\n" >&2;;
 esac
esac
```

```
done
$ chmod u+x datana1
$ datana1 eingabe wartmsg
```

Inhaltsanalyse fuer folgende Dateien:  
eingabe wartmsg

- 1) Wortstatistik
- 2) Durchschnittliche Zeilenlaenge
- 3) Zeilen-, Wort- und Zeichenanzahl
- 4) Verlassen

Ihre Wahl ? 1

```
1 0
6 1
3 2
1 60
1 an
1 angemeldet
1 banner
1 deine
1 do
1 done
1 du
5 echo
2 ein
1 eingegeben
1 exit
1 fi
2 gib
1 grep
1 hast
1 hat
1 if
1 mail
1 meldung
1 ne
2 read
1 reply
1 sich
1 sleep
1 then
1 until
1 usage
1 username
1 was
1 who
2 zeile
```

Ihre Wahl ?

- 1) Wortstatistik
- 2) Durchschnittliche Zeilenlaenge

```
3) Zeilen-, Wort- und Zeichenanzahl
4) Verlassen
Ihre Wahl ? 2
Durchschnittliche Zeilenlaenge: 15
Ihre Wahl ? 3
 3 10 64 eingabe
 14 45 216 wartmsg
 17 55 280 total
Ihre Wahl ? 4
$
```

### 5.16.8 time-Kommando

`time pipeline`

Die **ksh** führt die angegebene *pipeline*<sup>1</sup> aus und gibt auf die Standardfehlerausgabe drei Zeilen aus, welche in Minuten und Sekunden folgendes festlegen:

**real** vergangene Uhrzeit (*elapsed time*) bei der Ausführung der angegebenen *pipeline*.

**user** gebrauchte CPU-Zeit der *pipeline* im Benutzermodus.

**sys** gebrauchte CPU-Zeit der *pipeline* im Systemmodus (z.B. bei der Ausführung von Systemroutinen)

Der exit-Status des **time**-Kommandos ist der exit-Status der *pipeline*.

Werden beim Aufruf des **time**-Kommandos Ein-/Ausgabeumlenkungen angegeben, so beziehen sich diese auf die *pipeline* und nicht auf die Ein- oder Ausgaben des **time**-Kommandos.

**Beispiel** `$ time ls -l /bin /usr/bin | wc`

```
 400 3552 23248 [wc-Ausgabe]

real 0m5.56s
user 0m3.23s
sys 0m2.05s
$ time sleep 10

real 0m9.50s
user 0m0.00s
sys 0m0.06s
$ time pwd
/user1/egon/kshueb

real 0m0.00s
user 0m0.00s
sys 0m0.00s
$
```

---

1. Kann auch nur ein Kommando sein.

### 5.16.9 Funktionen

Auch in der Korn-Shell ist die Definition von Funktionen erlaubt. Die Syntax für eine Funktionsdefinition ist dabei zunächst wieder wie in der Bourne-Shell:

```
funktionsname() { kdoliste; }1
```

Neu in der **ksh** hinzugekommen ist eine alternative Angabe für Funktionsdefinitionen:

```
function funktionsname { kdoliste; }2
```

In beiden Fällen wird der Funktion *funktionsname* die *kdoliste* zugeordnet.

Der Aufruf von *funktionsname* bewirkt dann die Ausführung der zugeordneten *kdoliste*.

**Beispiel**

Die nachfolgende Funktion **verifik** kann für beliebige Ja / Nein-Eingaben aufgerufen werden. Sie liefert als Rückgabewert 0, wenn *Ja* oder *J* (klein oder groß geschrieben) eingegeben wird, und den Rückgabewert 1, wenn *N* oder *Nein* (klein oder groß geschrieben) eingegeben wird:

```
function verifik
{
typeset -l antwort
while true
do read -r antwort?"$1 ? " || return 1
 case $antwort in
 j|ja) return 0;;
 n|nein) return 1;;
 *) print "\7Nur j(a) oder n(ein) eingeben";;
 esac
done
}
```

Mit dieser Funktionsdefinition sind Aufrufe wie z.B. der folgende möglich:

```
while verifik 'Weitermachen j(a)/n(nein)'
do

done
```

- 
1. { und } sind dabei Schlüsselwörter. In Kapitel 5.16 wurden die Regeln angegeben, die bei Schlüsselwörtern zu beachten sind.
  2. { und } sind dabei Schlüsselwörter. In Kapitel 5.16 wurden die Regeln angegeben, die bei Schlüsselwörtern zu beachten sind.

## Eigenschaften von Shell-Funktionen

Nachfolgend sind die Eigenschaften von Funktionen in der Korn-Shell zusammengefaßt:

Da der Aufruf von Funktionen weitgehend dem Aufruf von Shell-Skripts entspricht, können auch in der **ksh** bei Funktionsaufrufen Argumente angegeben werden, welche dann der entsprechenden Funktion in Form von Positionsparametern zur Verfügung gestellt werden.

Jedoch werden die Positionsparametern nur für die Dauer der Funktions-Ausführung neu gesetzt; danach besitzen die Positionsparameter wieder ihre ursprünglichen Werte.

Anders als in der Bourne-Shell wird **\$0** bei Funktionsaufrufen mit dem Funktionsnamen besetzt.

### Beispiel

```
$ function ausgabe { print - "$0 $1 $2 $3"; }
$ set a b c
$ print - "$0 $1 $2 $3"
-ksh a b c
$ ausgabe Heute ist Freigabe-Termin
ausgabe Heute ist Freigabe-Termin
$ print - "$0 $1 $2 $3"
-ksh a b c
$
```

Eine Funktionsdefinition kann sich über mehrere Zeilen erstrecken. Die **ksh** gibt dann bei einer interaktiven Eingabe einer Funktionsdefinition auch wieder solange den Sekundär-Promptstring aus, bis die Definition mit der abschließenden **}** beendet wird.

Für die Wahl eines Funktionsnamens gelten die gleichen Regeln wie für die Wahl von Datei- und Variablenamen<sup>1</sup>.

Gleiche Funktions- und Shellvariablen-Namen sind erlaubt.

Beim Aufruf von Shell-Funktionen werden diese in der aktuellen Shell ausgeführt. Somit können sie verwendet werden, um Shell-Variablen in der momentan aktiven Shell zu verändern.

Funktionen sind nur in der Shell bekannt, in der sie definiert wurden.

In der **ksh** ist es jedoch möglich, Funktionen anderen **ksh**-Aufrufen bekannt zu machen. Dazu müssen die entsprechenden Funktionsdefinitionen in der *Environment*-Datei<sup>2</sup> angegeben werden. Bei jedem Aufruf der **ksh** werden dann die Funktionsdefinitionen aus dieser Datei gelesen und sind somit in der entsprechenden **ksh** bekannt.

1. Siehe Syntaxregeln für Bezeichner in Kapitel 3.

2. Über die vordefinierte Shell-Variable **ENV** festgelegt; siehe Kapitel 5.8.2.



Soll eine Funktion auch an Subshells exportiert werden, die nicht mit einem expliziten **ksh**-Aufruf (wie z.B. Shell-Skripts) erzeugt werden, so muß neben der Funktionsdefinition noch der entsprechende Funktionsname mit **typeset -xf** in der *Environment*-Datei definiert werden.

**Beispiel**

```
$ echo $ENV
/user1/kshueb/.ksh_env
$ cat $ENV
export PS1='!>$PWD: '

typeset -xf zaehle
function zaehle
{
 integer start=${1:-1} ende=${2:-10} step=${3:-1}
 integer i=start
 while ((i<=ende))
 do print - $i
 i=i+step
 done
}
$. $ENV [Lesen der Datei $ENV mit Punkt-Kommando]
212>/user1/egon/kshueb: PS1="$ "
$ cat sum_ungerad
typeset -i sum=0
for i in $(zaehle 1 ${1:-100} 2)
do sum=sum+$i
done
print $sum
$ chmod u+x sum_ungerad
$ sum_ungerad
2500
$ sum_ungerad 5
9
$ sum_ungerad 5000
6250000
$
```

In der **ksh** ist es möglich, funktionslokale Variablen zu deklarieren; dazu muß die entsprechende Variable innerhalb der Funktion mit **typeset** definiert werden. Das Modifizieren von lokalen Variablen in einer Funktion hat keinerlei Einfluß auf gleichnamige Variablen außerhalb der Funktion. Alle funktionslokalen Variablen sind nach der Ausführung der Funktion nicht mehr verfügbar.

Wird jedoch in einer Funktion eine Variable verwendet, welche dort nicht explizit mit **typeset** definiert wurde, so handelt es sich um eine globale Variable. Das bedeutet, daß jede Änderung Auswirkungen auf eine eventuell gleichlautende Variable außerhalb der Funktion hat. Zum anderen ist eine solche Variable auch nach dem Funktionsende noch vorhanden, selbst wenn sie zuvor noch nicht existierte.

**Beispiel**

```
$ name=anton
$ function e1 { typeset name=emil; print $name;}
$ e1
emil
$ print $name
anton
$ function e2 { name=emil; print $name;}
$ e2
emil
$ print $name
emil
$
```

Die builtin-Kommandos **dirs**, **pushd** und **popd** der C-Shell fehlen in der **ksh**. Eine mögliche einfache Realisierung dieser Kommandos als **ksh**-Funktionen wird hier gezeigt. Dazu sollten die nachfolgenden Funktionsdefinitionen in der Environment-Datei (**\$ENV**) eingetragen werden. Der Directory-Stack wird hierbei über die globale Variable **DS** realisiert:

```
$ cat $ENV
:
:

typeset -xf dirs pushd popd

function dirs
{
 print - "${PWD};$DS"
}

function pushd
{
 DS="$PWD;$DS"
 cd $1
 dirs
}

function popd
{
 if ["$DS"]
 then
 cd ${DS%%\;*}
 DS=${DS#*\;*}
 dirs
 else
 echo "Stack ist leer"
 fi
}

$. $ENV
243>/user1/egon/kshueb: PS1='$PWD: '
/user1/egon/kshueb: dirs
/user1/egon/kshueb;
/user1/egon/kshueb: pushd /bin
/bin;/user1/egon/kshueb;
```

```

/bin: pushd ../usr
/usr:/bin:/usr1/egon/kshueb;
/usr: popd
/bin:/usr1/egon/kshueb;
/bin: popd
/usr1/egon/kshueb;
/usr1/egon/kshueb: popd
Stack ist leer
/usr1/egon/kshueb: PS1="$ "
$

```

**ksh**-Funktionen können auch rekursiv aufgerufen werden.

#### Beispiel

Das nachfolgende komplexere Beispiel zeigt die Realisierung des berühmten Spiels »Türme von Hanoi« mit der **ksh**:

```

$ cat hanoi
#----- Funktionsdefinitionen -----
#-----
function hanoi
{
 integer zahl von nach on ov
 let zahl=$1-1
 (($zahl>0)) && hanoi $zahl $2 $4 $3
 von=$2
 nach=$4
 ((oben[nach]=oben[nach]-1))
 ((ov=oben[von]))
 ((on=oben[nach]))
 eval "turm$nach[$on]=\${turm$von[$ov]}"
 eval "turm$von[oben[von]]=\${leer}"
 ((oben[von]=oben[von]+1))
 ((zug=zug+1))
 print - "\n$zug.Zug: $von ---> $nach"
 for i in $zahlen
 do print - " ${turm1[$i]} ${turm2[$i]} ${turm3[$i]}"
 done
 print - " $basis $basis $basis"
 read
 (($zahl>0)) && hanoi $zahl $3 $2 $4
}

function init
{
 integer i=0
 while (((i=i+1) <= bloecke))
 do zahlen="$zahlen $i"
 done
 oben[1]=1
 ((oben[2]=bloecke+1))
 ((oben[3]=bloecke+1))
}

```

```

turm1[1]=" * "
turm1[2]=" *** "
turm1[3]=" ***** "
turm1[4]=" * "
turm1[5]=" ***** "
turm1[6]=" ***** "
turm1[7]=" ***** "
turm1[8]=" ***** "
turm1[9]=" ***** "
basis="-----"
leer=' '
for i in $zahlen
do turm2[$i]="$leer"
turm3[$i]="$leer"
done
}

#----- main -----
#-----

integer oben[3] zug=0 bloecke

bloecke=${1:-3}

case $bloecke in
[1-9]) init
hanoi $bloecke 1 2 3
exit 0;;
*) print "Blockanzahl muss im Intervall [1,5] liegen"
exit 1;;
esac

$ chmod u+x hanoi

$ hanoi 4

1.Zug: 1 ---> 2

 ***** *

[]

2.Zug: 1 ---> 3

 ***** * *****

[]

```

3. Zug: 2 ---> 3

\*\*\*\*\*  
\*\*\*\*\*

\*  
\*\*\*



4. Zug: 1 ---> 2

\*\*\*\*\*

\*\*\*\*\*

\*  
\*\*\*



5. Zug: 3 ---> 1

\*  
\*\*\*\*\*

\*\*\*\*\*

\*\*\*



6. Zug: 3 ---> 2

\*  
\*\*\*\*\*

\*\*\*  
\*\*\*\*\*



7. Zug: 1 ---> 2

\*\*\*\*\*

\*  
\*\*\*  
\*\*\*\*\*



8. Zug: 1 ---> 3

\*  
\*\*\*  
\*\*\*\*\*

\*\*\*\*\*



9. Zug: 2 ---> 3

\*\*\*  
\*\*\*\*\*

\*  
\*\*\*\*\*

10.Zug: 2 ---> 1

|       |       |            |
|-------|-------|------------|
| ***   | ***** | *<br>***** |
| ----- | ----- | -----      |

11.Zug: 3 ---> 1

|          |       |       |
|----------|-------|-------|
| *<br>*** | ***** | ***** |
| -----    | ----- | ----- |

12.Zug: 2 ---> 3

|          |       |                |
|----------|-------|----------------|
| *<br>*** |       | *****<br>***** |
| -----    | ----- | -----          |

13.Zug: 1 ---> 2

|       |       |                |
|-------|-------|----------------|
| ***   | *     | *****<br>***** |
| ----- | ----- | -----          |

14.Zug: 1 ---> 3

|       |       |                       |
|-------|-------|-----------------------|
|       |       | ***<br>*****<br>***** |
| ----- | ----- | -----                 |

15.Zug: 2 ---> 3

|       |       |                            |
|-------|-------|----------------------------|
|       |       | *<br>***<br>*****<br>***** |
| ----- | ----- | -----                      |

\$

Ein-/Ausgabeumlenkung ist sowohl bei der Definition als auch beim Aufruf einer Shell-Funktion erlaubt.

Der Aufruf des builtin-Kommandos

### **return** [*n*]

bewirkt, daß eine Funktion mit dem Rückgabewert (exit-Status) *n* verlassen wird. Ist *n* nicht angegeben, dann wird als Rückgabewert der exit-Status des zuletzt in dieser Funktion ausgeführten Kommandos verwendet.

#### Beispiel

Die Realisierung des Kommandos **basename** als Funktion könnte wie folgt aussehen:

```
$ cat bname
function basename
{ case $# in
 1) ;;
 2) eval set \${1%$2};;
 *) print - "usage: $0 pfadname [suffix]"
 return 1;;
 esac
 print - ${1##*/}
 return 0
}
$. bname
$ basename $HOME
egon
$ ls *.c
addiere.c
$ basename *.c
addiere.c
$ basename *.c .c
addiere
$
```

Builtin-Kommandos<sup>1</sup> können nicht durch eine Funktionsdefinition ersetzt werden, andere Kommandos dagegen sehr wohl.

Jedoch ist es über einen kleinen Trick, bei dem eine Kombination von Alias- und Funktionsdefinitionen angegeben wird, doch möglich, ein builtin-Kommando durch eine Funktion zu ersetzen<sup>2</sup>.

Die in der gerade aktiven **ksh** definierten Shell-Funktionen können mit dem Kommando

**typeset -f** (ohne Angabe von Argumenten)

oder dem Aufruf des vordefinierten Alias

### **functions**

angezeigt werden.

---

1. Im Kapitel 5.24.5 sind alle builtin-Kommandos der **ksh** angegeben.

2. Siehe Kapitel 5.17.

Mit

**unset -f** *funktionsname*

kann die Definition einer Shell-Funktion wieder aufgehoben werden.

Beim Aufruf von Funktionen werden diese immer als solche erkannt, selbst, wenn Quoting verwendet wird.

**Hinweise**

Bei der Erstellung von Funktionen sollte folgendes beachtet werden:

- ▶ Auf globale Variablen sollte innerhalb von Funktionen nur dann zugegriffen werden, wenn dies unbedingt erforderlich ist. Ansonsten sollten funktions-lokale Variablen verwendet werden.
- ▶ Da Funktionen in der aktuellen Shell ausgeführt werden, bewirkt ein Wechsel des Directorys in einer Funktion, daß dies nach Rückkehr aus der Funktion auch das neue Working-Directory der Shell ist, wenn in der Funktion nicht explizit zum ursprünglichen Working-Directory zurückgekehrt wird.
- ▶ Ein weiterer Punkt, den es zu beachten gilt, weil Funktionen in der aktuellen Shell ausgeführt werden, ist die gleiche PID der aktuellen Shell und der Funktion. Dies kann beim Kreieren von temporären Dateien durch Anhängen von \$\$ (zum Beispiel *tmpname=/tmp/temp\$\$*) an den Namen zu Problemen der Art führen, daß die aktuelle Shell und die Funktion unbeabsichtigt die gleiche temporäre Datei benutzen. Der Entwickler sollte also über funktions-eigene Dateinamen für temporäre Dateien sicherstellen, daß solche Konflikte vermieden werden.
- ▶ Auf **ksh**-Versionen, die vor dem 3.6.1986 freigegeben wurden, waren alle Signalbehandlungs-Routinen (außer für **EXIT**<sup>1</sup>), die mit **trap** eingerichtet wurden, sowohl für die Funktion als auch für die entsprechende Shell aktiv.
- ▶ Auf **ksh**-Versionen nach dem 3.6.1986 können für Funktionen und die aktuelle Shell unterschiedliche Signal-Handler installiert werden. Ein innerhalb der Funktion mit **trap** installierter Signal-Handler gilt deshalb nur für die Dauer der Funktionsausführung. Nach der Rückkehr aus der Funktion werden automatisch wieder die Signal-Handler der aktuellen **ksh** aktiviert.

**Autoload Funktionen**

Eine Funktion, die erst dann definiert wird, wenn sie das erstmal aufgerufen wird, wird als *autoload-Funktion* bezeichnet.

Autoload-Funktionen haben den Vorteil, daß die **ksh** keine Zeit mit dem Lesen von Funktionsdefinitionen vergeudet, die niemals aufgerufen werden.

---

1. Siehe Kapitel 5.22.



Auf **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, können auto-load-Funktionen mit dem vordefinierten Alias **autoload**<sup>1</sup> oder mit **typeset -fu** deklariert werden.

Beim ersten Aufruf einer **autoload**-Funktion durchsucht die **ksh** die in der Shellvariablen **FPATH** enthaltenen Directories, um nach einem Dateinamen zu suchen, der der Name dieser Funktion ist. Findet die **ksh** eine gleichnamige Datei, so liest sie zunächst den Inhalt dieser Datei (in der aktiven Shell) und führt dann die entsprechende Funktion aus.

**Beispiel**

```
$ cd
$ pwd
/user1/egon
$ mkdir flib
$ cd flib
$ ls
$ cat isnum
-- isnum -----
Rueckgabewert: 0 (TRUE), wenn erstes Argument
nur aus Ziffern besteht.
1 (FALSE), ansonsten
function isnum
{ typeset para="$1" zahl=1 # zahl ist boole'sche Var.
 while true
 do case $para in
 [0-9]*) para=${para#?}
 zahl=0;;
 "") return $zahl;;
 *) return 1;;
 esac
 done
}
$ FPATH=$HOME/flib [am besten in .profile eintragen]
$ cd $HOME/kshueb
$ autoload isnum [am besten in $ENV ($HOME/.ksh_env)
 eintragen]
$ typeset -f | grep isnum [keine Funktion isnum vorhanden]
$ isnum 7289352
$ print $?
0 [7289352 ist eine Zahl]
$ typeset -f | grep isnum
function isnum [nun ist eine Funktion isnum vorhanden]
$ if isnum 089-1234556
> then
> print "Zahl"
> else
> print "keine Zahl"
> fi
keine Zahl
$
```

---

1. Siehe Kapitel 5.17.

Auf **ksh**-Versionen, die am 3.6.1986 oder früher freigegeben wurden, wird der **autoload**-Mechanismus nicht eigens angeboten; dieser kann aber mit der Definition eines Alias, dessen Name der Name einer Funktion ist, nachgebildet werden<sup>1</sup>.

## 5.17 Der Alias-Mechanismus

Mit dem Alias-Mechanismus der **ksh** ist es möglich, an einen ganzen oder auch unvollständigen Kommandoaufruf einen Kurznamen, auch *Alias* genannt, zu vergeben.

### 5.17.1 Das builtin-Kommando alias

Mit dem builtin-Kommando **alias** können entweder Aliase definiert oder aber momentan definierte Aliase angezeigt werden:

**alias** [-**tx**] [*name*[=*wert*]]<sup>2</sup>

Sind keine Argumente der Form *name*[=*wert*] angegeben, so gibt die **ksh** eine Liste der momentan definierten Aliase auf die Standardausgabe aus, und zwar pro Zeile ein Alias in der Form *name*=*wert*.

Ist **-t** und / oder **-x** angegeben, dann werden nur die Aliase mit den entsprechenden Attributen (siehe unten) angezeigt.

Wird *name*=*wert* angegeben, so wird ein Alias *name* mit dem angegebenen *wert* neu definiert. Vorherige Definitionen von *name* werden dabei überschrieben. Für Aliasnamen gilt, daß das erste Zeichen kein Metazeichen sein darf und die restlichen Zeichen Buchstaben oder Ziffern sein müssen.

#### Optionen

| Option    | Bedeutung                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-t</b> | ( <i>tracked</i> ) Markieren eines Alias bzw. Auflisten der markierten Aliase.<br>Der Wert eines mit <b>-t</b> markierten Alias ist der absolute Pfadname des Programms <i>name</i> ; ein so markiertes Alias verliert diesen Wert, wenn die Variable <b>PATH</b> verändert wird, wobei jedoch das Alias selbst mit <b>-t</b> markiert bleibt. (siehe unten: Tracked Alias). |
| <b>-x</b> | ( <i>export</i> ) Markieren eines Alias zum Export bzw. Auflisten der zum Export markierten Aliase.                                                                                                                                                                                                                                                                          |

1. Siehe Kapitel 5.17.

2. Vor und nach dem Gleichheitszeichen darf kein Trennzeichen (wie z.B. Leerzeichen) angegeben sein.

| Option | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | <p>Ein Alias kann jedoch nur an Subshells exportiert werden, die durch den Aufruf eines Shellskripts (ohne <b>ksh</b>) erzeugt werden.</p> <p>Wird eine neue Korn-Shell mit dem Aufruf von <b>ksh</b> gestartet, so werden die mit <b>-x</b> markierten Aliase nicht automatisch dorthin exportiert. Um dies zu erreichen, müßten die entsprechenden Alias-Definitionen in der Environment-Datei angegeben sein, damit sie in die neue <b>ksh</b> vererbt werden.</p> |

Wird nur ein *name* (ohne die Zuweisung eines *wert*) angegeben, so sind folgende Fälle zu unterscheiden:

- ▶ keine Angabe von **-t** oder **-x**:  
*name* und *wert* des Alias *name* wird angezeigt.
- ▶ Angabe von **-x**:  
Alias *name* wird für den Export markiert.
- ▶ Angabe von **-t**:  
Alias *name* wird mit **-t** markiert. Zusätzlich wird noch der Wert dieses Alias auf den absoluten Pfadnamen des entsprechenden Programms *name* gesetzt; dazu werden die Directories in der Variablen **PATH** durchsucht (siehe unten: Tracked Aliase).

*wert* kann ein beliebiger Text (einer Kommandozeile) sein. Wenn das letzte Zeichen von *wert* ein Leer- oder Tabulatorzeichen ist, dann prüft die **ksh** bei der Alias-Substitution auch noch das dem Alias folgende Wort daraufhin, ob es sich dabei um ein Alias handelt, das substituiert werden muß. Allgemein sollte immer dann ein Alias-Text mit einem Leer- oder Tabulatorzeichen enden, wenn einem Alias ein weiteres Alias folgen kann.

Wenn ein Alias-Text immer dann zu expandieren ist, wenn auf das entsprechende Alias zugegriffen wird, so muß der Alias-Text mit **'..'** geklammert werden, ansonsten wird der Alias-Text nur einmal bei der **alias**-Definition expandiert.

In einem Alias-Text kann der eigene Alias-Name verwendet werden; es handelt sich dann nicht um das Alias, sondern um das entsprechende Kommando:

```
alias ls='ls -CF '
```

```
Beispiel $ alias dir='ls '
$ alias type='cat '
$ alias -x md='mkdir ' rd='rmdir '
$ alias del='rm '
$ alias
autoload=typeset -fu [Neben den explizit definierten Aliase enthält]
del=rm [diese Liste auch vordefinierte Aliase (siehe unten)]
dir=ls
false=let 0
functions=typeset -f
hash=alias -t -
history=fc -l
integer=typeset -i
md=mkdir
nohup=nohup
r=fc -e -
rd=rmdir
stop=kill -STOP
suspend=kill -STOP $$
true=:
type=cat
$ dir b*
basename
bname
$ type eingabe
echo "Gib was ein:"
read
echo "Du hast ---$REPLY--- eingegeben"
$ x=1
$ alias druck1='print $x' druck2="print $x"
$ x=5
$ druck1
5
$ druck2
1
$ print $OLDPWD
/user1/egon
$ alias alt="cd $OLDPWD"
$ alias alt
alt=cd /user1/egon
$ cd /usr
$ alt
$ pwd
/user1/egon
$ cd kshueb
$ print $OLDPWD
/user1/egon
$ alias alt='cd $OLDPWD' [statt mit "." nun mit '..' geklammert]
$ alias alt
alt=cd $OLDPWD
$ cd /usr
```

```
$ alt ↵
$ pwd ↵
/user1/egon/kshueb
$
```

Der exit-Status des **alias**-Kommandos ist 0 (erfolgreich), wenn alle angegebenen *name(n)* Aliase sind oder wenn mit einem **alias**-Aufruf Attribute gesetzt werden. Ansonsten ist der exit-Status von **alias** die Zahl der *name(n)*, die keine Aliase sind.

### 5.17.2 Löschen eines Alias

Mit dem builtin-Kommando **unalias** können zuvor definierte Aliase gelöscht werden:

**unalias** *name(n)*

Der exit-Status von **unalias** ist 0 (erfolgreich), wenn alle angegebenen *name(n)* Aliase sind. Ansonsten ist der exit-Status die Anzahl von Namen, die keine Aliase sind.

### 5.17.3 Alias-Substitution

Für jedes einfache Kommando prüft die **ksh** das erste Wort (Kommandoname), um dieses eventuell als Alias zu klassifizieren. Wenn keinerlei Quoting in diesem Wort vorkommt und das Wort ein erlaubter Alias-Name ist, der nicht mit **-t** markiert wurde, dann setzt die **ksh** für den Alias, falls definiert, den entsprechenden Alias-Text ein. Falls im Alias-Text der Alias-Name selbst nochmals vorkommt, so wird dieser nicht erneut substituiert. Kommt im Alias-Text dagegen ein anderer Alias-Name vor, so wird auf **ksh**-Versionen nach dem 3.6.1986 auch dieser substituiert.

Normalerweise wird die Alias-Substitution nur für das erste Wort eines einfachen Kommandos durchgeführt. Endet jedoch ein Alias-Text mit einem Leer- oder Tabulatorzeichen, dann wird auch noch für das nachfolgende Wort Alias-Substitution durchgeführt.

#### Beispiel

```
$ alias druck=print ↵
$ alias text="Hallo egon" ↵
$ druck text ↵
text
$ alias drucke='print ' ↵
$ drucke text ↵
Hallo egon
$ alias w=wc ↵
$ drucke text | w ↵
1 2 11
$ alias pdir='basename `pwd`' ↵
$ pdir ↵
```

```
kshueb
$ cd /usr/bin
$ pdir
bin
$ alt
$ pwd
/user1/egon/kshueb
$
```

### 5.17.4 Tracked Aliase

Wie die Bourne-Shell, so verfügt auch die **ksh** über ein sogenanntes *Kommando-Tracking*: Wenn ein Kommando aufgerufen wird, das kein builtin-Kommando und keine definierte Funktion ist, so durchsucht die Shell die in **PATH** angegebenen Directories nach diesen Kommandonamen. Um beim nächsten Aufruf des gleichen Kommandos nicht wieder diesen zeitaufwendigen Suchvorgang wiederholen zu müssen, merken sich beide Shells den Pfad des Kommandos. Während die Bourne-Shell sich die bereits gefundenen Pfadnamen von Kommandos in einer internen hashing-Tabelle speichert, verwendet die **ksh** ein anderes Verfahren:

Das erstmal, wenn ein Kommando aufgerufen wird, kreiert die **ksh** ein sogenanntes *Tracked Alias* mit dem gleichen Namen wie das betreffende Kommando und weist diesem Alias den absoluten Pfadnamen des Kommandos als Wert zu.

Während in der Bourne-Shell das Kommando-Tracking immer eingeschaltet ist, kann es in der **ksh** vom Benutzer ein- und ausgeschaltet werden. Die Voreinstellung ist, daß Kommando-Tracking nicht eingeschaltet ist. Mit

#### **set -o trackall**

kann das Kommando-Tracking ein- und mit

#### **set +o trackall**

ausgeschaltet werden.

Mit dem Aufruf

#### **alias -t**

können alle momentan definierten Tracked Aliase angezeigt werden.

```
Beispiel $ wc eingabe
 3 10 64 eingabe
$ alias -t
$ set -o trackall
$ wc eingabe
 3 10 64 eingabe
$ alias -t
wc=/bin/wc
$ ls b*
```

```
basename
bname
$ alias -t
ls=/bin/ls
wc=/bin/wc
$
```

Soll das Kommando-Tracking immer automatisch eingeschaltet werden, so sollte

### **set -o trackall**

in der Environment-Datei **\$ENV** eingetragen werden.

Das Kommando-Tracking hat den Vorteil, daß der zeitaufwendige Suchvorgang für häufig benutzte Kommandos wie **ls**, **cat**, **vi** usw. nur einmal beim ersten Aufruf durchgeführt wird, und somit diese Kommandos bei nachfolgenden Aufrufen wesentlich schneller starten. Dies ist insbesondere für Kommandos sehr nützlich, die in Schleifen aufgerufen werden.

Ein Benutzer kann auch explizit Tracked Aliase kreieren, ohne daß er das entsprechende Kommando aufruft:

### **alias -t kdoname(n)**

Benutzer, die auch den zeitaufwendigen Suchvorgang beim ersten Aufruf bestimmter Kommandos unterbinden möchten, sollten deshalb den Aufruf

### **alias -xt kdoname(n)**

für die entsprechenden *kdoname(n)* in *.profile* eintragen.

```
$ alias -t
ls=/bin/ls
wc=/bin/wc
$ alias -t cat vi
cat=/bin/cat
ls=/bin/ls
vi=/usr/bin/vi
wc=/bin/wc
$ alias -t zufall
$ alias -t
cat=/bin/cat
ls=/bin/ls
vi=/usr/bin/vi
wc=/bin/wc
zufall=/user1/egon/kshueb/./zufall
$
```

Jedesmal, wenn die **PATH**-Variable neu gesetzt wird, werden die Werte aller Tracked Aliase gelöscht; die Aliase selbst bleiben jedoch bestehen und besitzen weiter das *tracked*-Attribut.

```

$ PATH=$PATH
$ alias -t [keine tracked Aliase mit Wert vorhanden]
$ set +o trackall [Kommando-Tracking ausschalten]
$ zufall
1.Zufallszahl: 10554
2.Zufallszahl: 1407
Zufahl aus [1,100]: 89
$ alias -t
zufall=/user1/egon/kshueb/./zufall
$ wc eingabe
 3 10 64 eingabe
$ alias -t
wc=/bin/wc
zufall=/user1/egon/kshueb/./zufall
$ file abc
abc: empty
$ alias -t
wc=/bin/wc
zufall=/user1/egon/kshueb/./zufall
$

```

### 5.17.5 Vordefinierte Aliase

Folgende Aliase werden von der **ksh** automatisch vordefiniert:

```

autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=:
type='whence -v'

```

Diese vordefinierten Aliase kann der Benutzer zwar ändern oder löschen. Dies ist jedoch nicht ratsam, da dadurch der vom Hersteller vorgegebene Standard unnötigerweise wieder aufgehoben und eine verminderte Portabilität der eigenen **ksh**-Skripts nach sich ziehen würde.

### 5.17.6 Funktionen und Aliase

Im vorherigen Kapitel wurde darauf hingewiesen, daß builtin-Kommandos nicht durch Funktionen ersetzt werden können. Diese Einschränkung gilt zwar im allgemeinen, da die **ksh** bei einem Kommandoaufruf zuerst prüft, ob es ein builtin-Kommando ist, und erst dann nach einer Funktion mit diesem Namen

- 
1. **file** wird kein Tracked Alias, da Kommando-Tracking ausgeschaltet und **file** anders als **wc** und **zufall** nicht das tracked-Attribut besitzt.



sucht. Da die **ksh** jedoch vor der Prüfung auf ein builtin-Kommando die Prüfung auf ein Alias durchführt, kann diese Einschränkung durch eine geeignete Kombination von Aliasen und Funktionen umgangen werden.

Um ein builtin-Kommando durch eine Funktion zu ersetzen, muß ein Alias mit dem Namen des zu ersetzenden builtin-Kommandos definiert werden. Als Alias-Wert ist dabei der entsprechende Funktionsname anzugeben. Bei der Definition der betreffenden Funktion ist dann nur noch darauf zu achten, daß der Aufruf des wirklichen builtin-Kommandos (falls benötigt) mit Quoting angegeben wird, um eine Alias-Substitution für diesen Namen zu unterbinden.

**Beispiel**

```
$ alias cd=_cd
$ cat mein_cd
function _cd
{ \cd "$@" # Quoting, um rekursiven Aufruf
 # der Funktion cd zu vermeiden.
 print "Directorywechsel: $OLDPWD ---> $PWD"
}
$. mein_cd
$ cd /usr
Directorywechsel: /user1/egon/kshueb ---> /usr
$ pwd
/usr
$ cd
Directorywechsel: /usr ---> /user1/egon
$ pwd
/user1/egon
$ cd kshueb
Directorywechsel: /user1/egon ---> /user1/egon/kshueb
$ pwd
/user1/egon/kshueb
$ unalias cd [um richtiges cd wieder verwenden zu können]
$
```

Um **autoload**-Funktionen auf **ksh**-Versionen vor dem 3.6.1986 nachzubilden, muß ein Alias-Name mit dem Namen der entsprechenden **autoload**-Funktion definiert werden. Der Wert dieses Aliases muß ein Punkt-Kommando mit einem Argument sein. Das Argument muß dabei der Pfadname der Datei sein, welche die Funktionsdefinition enthält; nach dem Dateinamen muß ein Semikolon gefolgt vom Funktionsnamen angegeben werden. In der Funktionsdefinitions-Datei muß der entsprechende Alias-Name zunächst mit **unalias** entfernt werden, bevor dieser Name dann als Funktionsname definiert wird.

```
Beispiel $ alias isprim='. $HOME/flib/isprim; isprim'↵
$ cat $HOME/flib/isprim↵
unalias isprim # Loeschen des Alias isprim

-- isprim -----
#
Rueckgabewert: 0 (TRUE), wenn erstes Argument
eine Primzahl ist
1 (FALSE), ansonsten
function isprim
{
 set -- $(factor $1 2>/dev/null)
 ((rueckgabe=$((rueckgabe=0)-2))) # wenn Primzahl, dann liefert
 # factor 2 Zahlen
 return $rueckgabe
}
$ typeset -f | grep isprim↵ [Funktion isprim nicht vorhanden]
$ if isprim 123457↵
> then echo Primzahl↵
> else echo keine Primzahl↵
> fi↵
Primzahl
$ functions | grep isprim↵
function isprim [Funktion isprim nun vorhanden]
$ if isprim 12345678↵
> then echo Primzahl↵
> else echo keine Primzahl↵
> fi↵
keine Primzahl
$
```

Es könnte auch eine ganze Bibliothek von verwandten Funktionen geladen werden, wenn eine Funktion aus dieser Bibliothek das erstmal aufgerufen wird. Dazu müßte für jeden in der Bibliothek vorhandenen Funktionsnamen ein symbolischer Link auf die Bibliotheks-Datei gelegt werden, wie z.B.

**In** *funktionsname1* *\$HOME/flib/mathlib*

**In** *funktionsname2* *\$HOME/flib/mathlib*

....

Mit der Kombination von Aliassen und Funktionen ist es auch möglich, neue Sprachkonstrukte in der jeweiligen **ksh**-Umgebung einzuführen.

**Beispiel** `$ cat $HOME/flib/_from`

```
function _from
{
 typeset var="${1%=%}"
 integer incr="${5:-1}" "$1"
 while (($var<=$3))
 do
 _repeat
 let $var=$var+incr
 done
}

$ alias repeat='function _repeat {'
$ alias from='}; _from'
$
```

Ein Beispiel für die Verwendung der doch recht komplexen Definitionen könnte sein:

```
$ repeat
> { ((x=i*i))
> print "$i $x"
> }
> from i=1 to 13 step 3
1 1
4 16
7 49
10 100
13 169
$
```

**Hinweise**

- ▶ Da Alias-Substitution erst durchgeführt wird, wenn alle Schlüsselwörter einer Kommandozeile bearbeitet wurden, wird ein Alias-Name mit dem gleichen Namen wie ein Schlüsselwort niemals von der **ksh** als Alias erkannt werden.
- ▶ Aliase mit Namen der Form \_buchstabe (z.B. \_a, \_b, usw.) definieren Makros für die beiden builtin-Editoren<sup>2</sup> **vi** und **emacs**.

---

1. Diese beiden Zeilen könnten in die Environment-Datei **\$ENV** eingetragen werden. Um \_from explizit als **autoload**-Funktion zu kennzeichnen, sollte zusätzlich noch **autoload \_from** in der Environment-Datei **\$ENV** eingetragen werden.

2. Siehe Kapitel 5.19.

## 5.18 Die Tilde-Expandierung

Nachdem die **ksh** die Alias-Substitution durchgeführt hat, prüft sie jedes Wort in der Kommandozeile, ob es eventuell mit **~** (engl. *tilde*) beginnt. Ist ein solches Wort vorhanden, dann führt sie für dieses Wort vom Anfang bis zum nächsten Vorkommen von **/** bzw. bis zum Ende die sogenannte *Tilde-Expandierung* durch:

| Text                 | Tilde-Expandierung (wird ersetzt durch)     |
|----------------------|---------------------------------------------|
| <b>~</b>             | <b>\$HOME</b>                               |
| <b>~+</b>            | <b>\$PWD</b>                                |
| <b>~-</b>            | <b>\$OLDPWD</b>                             |
| <b>~loginname</b>    | <b>\$HOME</b> vom Benutzer <i>loginname</i> |
| <b>~anderer-text</b> | keine Ersetzung                             |

### Beispiel

```
$ pwd
/user1/egon/kshueb
$ print ~
/user1/egon
$ print ~emil
/user1/emil
$ cd ~emil/bin
$ pwd
/user1/emil/bin
$ cd ~-
$ pwd
/user1/egon/kshueb
$ d=$HOME/.profile s=~emil
$ print ${d}_datei $s/bin
/user1/egon/.profile_datei /user1/emil/bin
$
```

### Hinweis

Bei Variablenzuweisungen führt die **ksh** Tilde-Expandierung durch, wenn **~** am Anfang des entsprechenden Werts (unmittelbar nach dem Gleichheitszeichen) angegeben ist. **ksh**-Versionen vom 3.6.1986 und frühere führten Tilde-Expandierungen sogar nach jedem **:** (Doppelpunkt) in einem Wert einer Variablenzuweisung durch.

## 5.19 Der History-Mechanismus

Die **ksh** merkt sich die eingegebenen Kommandos in der sogenannten History-Datei. Das hier vorgestellte builtin-Kommando **fc** und die beiden im nächsten Kapitel vorgestellten builtin-Editoren **vi** und **emacs** ermöglichen es, Kommandos aus der History-Datei zu holen und zu editieren.

### 5.19.1 Allgemeines

Um sich die Kommandos aus der History-Datei anzeigen zu lassen, steht das vordefinierte Alias **history**<sup>1</sup> zur Verfügung.

Falls die Option **nolog**<sup>2</sup> nicht gesetzt ist, dann werden auch Funktionsdefinitionen in der History-Datei mit aufgenommen.

Die **ksh** öffnet die History-Datei, wenn sie eine Funktionsdefinition liest oder nachdem die Environment-Datei gelesen wurde, je nachdem, was zuerst eintrifft.

Der Name für die History-Datei wird über die Shellvariable **HISTFILE**<sup>3</sup> festgelegt.

Die **ksh** schreibt automatisch jede eingegebene Kommandozeile an das Ende der History-Datei. Jedoch können mit **read** oder **print** auch Kommandozeilen am Ende der History-Datei untergebracht werden, ohne daß sie ausgeführt werden:

**read -s kdozeile**

**print -s kdozeile**

Um beispielsweise den Inhalt der **PATH**-Variablen zu editieren, bevor man diese neu setzt, müßte zunächst

**print -s PATH=\$PATH**

aufgerufen werden. Danach könnte dann diese neu in die History-Datei eingetragene Zeile editiert werden, bevor sie der **ksh** zur Ausführung übergeben wird.

Über die Shellvariable **HISTSIZE**<sup>4</sup> kann festgelegt werden, wie viele der letzten Kommandozeilen maximal in der History-Datei aufzuheben sind.

Am Ende einer login-Sitzung wird die History-Datei nicht von der **ksh** gelöscht, so daß Kommandos von einer früheren Sitzung auch noch in einer späteren Sitzung wieder verwendet werden können.

### 5.19.2 Das builtin-Kommando fc

Um Kommandozeilen aus der History-Datei anzuzeigen oder zur Ausführung zu bringen, steht neben den beiden im nächsten Kapitel vorgestellten builtin-Editoren **vi** und **emacs** auch noch das builtin-Kommando **fc** (**fix command**) zur Verfügung. **fc** kann auf zwei verschiedene Arten aufgerufen werden, da es für zwei unterschiedliche Anwendungsfälle benutzt wird:

1. Siehe Kapitel 5.17: **history** ist vordefiniert als: **alias history='fc -l'**; das Kommando **fc** wird weiter unten in diesem Kapitel vorgestellt.
2. Nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.
3. Voreinstellung: **HISTFILE=\$HOME/.sh\_history**.
4. Voreinstellung: **HISTSIZE=128**.

- ▶ Erneute Ausführung einer zuvor eingegebenen Kommandozeile:  
`fc -e - [alt=neu] [kommando]`
- ▶ Editieren bzw. Auflisten der Kommandozeilen aus der History-Datei:  
`fc [-e editor] [-n1r] [von [bis]]`

### Erneute Ausführung einer zuvor eingegebenen Kommandozeile

Die Aufrufform

`fc -e - [alt=neu] [kommando]`

wird verwendet, wenn eine zuvor eingegebene Kommandozeile unverändert oder leicht abgeändert erneut ausgeführt werden soll. Die **ksh** gibt dabei die entsprechende Kommandozeile vor der Ausführung nochmals aus.

Wenn *alt=neu* angegeben ist, so wird in der entsprechenden Kommandozeile vor der Ausführung der String *alt* durch *neu* ersetzt.

*kommando* legt die auszuführende Kommandozeile fest. Für *kommando* kann dabei folgendes angegeben werden:

| Angabe               | Wirkung                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------|
| <i>positive Zahl</i> | legt die Nummer der entsprechenden Zeile aus der History-Datei fest.                      |
| <i>negative Zahl</i> | wird von der momentanen Kommandonummer subtrahiert.                                       |
| <i>string</i>        | wählt aus den vorherigen Kommandozeilen die letzte aus, welche mit <i>string</i> beginnt. |
| keine Angabe         | vorherige Kommandozeile.                                                                  |

Anstelle des obigen Aufrufs kann auch das vordefinierte Alias **r**<sup>1</sup> benutzt werden; auch dabei dürfen die Argumente *alt=neu* und *kommando* angegeben werden.

Der exit-Status dieser Aufrufform ist der exit-Status des Kommandos, welches durch **fc** zur Ausführung gebracht wurde.

#### Beispiel

```
$ HISTSIZE=100
$ echo $HISTSIZE
100
$ ls b*
basename
bname
$ r [Letztes Kommando wiederholen]
ls b*
basename
```

1. Siehe Kapitel 5.17: **r** ist vordefiniert als **alias r='fc -e -'**.

```

bname
$ r b=c [↵] [Letztes Kdo wiederholen; dabei b durch c ersetzen]
ls c*
ccvi
cph
$ r e [↵] [Letztes Kdo wiederholen, das mit e beginnt]
echo $HISTSIZE
100
$ r -5 [↵] [5.letztes Kdo wiederholen]
echo $HISTSIZE
100
$ r -5 [↵] [5.letztes Kdo wiederholen]
ls b*
basename
bname
$ r e echo=print [↵] [Letztes Kdo wiederholen, das mit e beginnt]
print $HISTSIZE [dabei echo durch print ersetzen]
100
$ r H HISTSIZE=a [↵] [Letztes Kdo wiederholen, das mit H beginnt]
a=100 [dabei HISTSIZE durch a ersetzen]
$ fc -e - ls ls=file [↵]
file b*
basename: commands text
bname: program text
$

```

## Editieren bzw. Auflisten der Kommandozeilen aus der History-Datei

### Die Aufrufform

```
fc [-e editor] [-nlr] [von [bis]]
```

wird verwendet, wenn

- ▶ der Inhalt der History-Datei aufzulisten ist:

**fc -l**      oder alternativ  
**history** (vordefiniertes Alias<sup>1</sup>)

- ▶ zuvor eingegebene Kommandozeilen vor ihrer Ausführung editiert werden sollen.

Im zweiten Fall darf die Option **-l** nicht angegeben werden. Die mit diesem Aufruf ausgewählten Kommandozeilen aus der History-Datei werden in eine temporäre Datei geschrieben, für die dann ein Editor aufgerufen wird. Als Editor wird dabei der nach **-e** angegebene *editor* verwendet. Fehlt diese Angabe, so wird der Editor verwendet, der in der Variablen **FCEDIT**<sup>2</sup> angegeben ist. Wurde diese

---

1. Siehe Kapitel 5.17: **history** ist vordefiniert als **alias history='fc -l'**.  
2. Siehe auch Kapitel 5.8.2.

Variable nicht explizit vom Benutzer gesetzt, so gilt ihre Voreinstellung: **FCE-DIT=/bin/ed**.

Nachdem die Editorsitzung beendet ist, liest die **ksh** die so editierten Kommandozeilen aus der temporären Datei, listet sie auf und führt sie dann alle nacheinander aus.

*von* und *bis* legen bei diesem Aufruf den Bereich der zu editierenden Kommandozeilen aus der History-Datei fest. Für *von* und *bis* kann dabei folgendes angegeben werden:

| Angabe                                    | Wirkung                                                                                                                                                      |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>positive Zahl</i>                      | legt die Nummer der entsprechenden Zeile aus der History-Datei fest                                                                                          |
| <i>negative Zahl</i>                      | wird von der momentanen Kommandonummer subtrahiert                                                                                                           |
| <i>string</i>                             | wählt aus den vorherigen Kommandozeilen die letzte aus, welche mit <i>string</i> beginnt.                                                                    |
| <i>bis</i> nicht angegeben                | für <i>bis</i> wird der Wert von <i>von</i> genommen.                                                                                                        |
| <i>von</i> und <i>bis</i> nicht angegeben | Es werden die folgenden default-Werte verwendet:<br><i>von</i> <b>-16</b> , wenn Option <b>-l</b> angegeben ist, ansonsten <b>-1</b><br><i>bis</i> <b>-1</b> |

## Optionen

Die einzelnen Optionen haben folgende Auswirkungen:

| Option    | Bedeutung                                                                                                        |
|-----------|------------------------------------------------------------------------------------------------------------------|
| <b>-l</b> | ( <b>list</b> ) Anzeigen von Kommandos aus der History-Datei.                                                    |
| <b>-n</b> | ( <b>no number</b> ) Bei der Ausgabe der Kommandozeilen aus der History-Datei die Kommandonummer nicht anzeigen. |
| <b>-r</b> | ( <b>reverse</b> ) Reihenfolge der Kommandos aus der History-Datei umdrehen.                                     |

## exit-Status

Der exit-Status dieser Aufrufform ist 1 (nicht erfolgreich), wenn ungültige Argumente beim Aufruf angegeben wurden, ansonsten wird unterschieden: Ist die Option **-l** angegeben, so ist der exit-Status 0 (erfolgreich), andernfalls ist der exit-Status der exit-Wert des letzten von **fc** zur Ausführung gebrachten Kommandos.



**Beispiel**

```
$ history -5[↵]
71 echo $HISTSIZE
72 ls b*
73 print $HISTSIZE
74 a=100
75 file b*
76 history -5
$ history 70 74[↵]
70 echo $HISTSIZE
71 echo $HISTSIZE
72 ls b*
73 print $HISTSIZE
74 a=100
$ history 72[↵] [ab 72 alle ausgeben]
72 ls b*
73 print $HISTSIZE
74 a=100
75 file b*
76 history -5
77 history 70 74
78 history 72
$ fc[↵] [ed für letzte Kommandozeile aufrufen]
12 [ed-Meldung: 12 Zeichen gelesen]
P[↵] [ed-Promptzeichen einschalten]
*,p[↵] [Dateiinhalt anzeigen]
history 72
*s/72/69[↵] [Ersetze 72 durch 69]
history 69
*w[↵] [Dateiinhalt sichern]
12 [ed-Meldung: 12 Zeichen gesichert]
*q[↵] [ed verlassen]
history 69 [Anzeigen des Kdos, das ausgeführt wird]
69 ls c*
70 echo $HISTSIZE
71 echo $HISTSIZE
72 ls b*
73 print $HISTSIZE
74 a=100
75 file b*
76 history -5
77 history 70 74
78 history 72
79 history 69
$ fc -e vi 72 74[↵] [Kdos 72 bis 74 mit vi editieren]
ls b* [im vi werden die entspr. 3 Kdos angezeigt]
print $HISTSIZE
a=100

:
[Mit vi-Kdos nun folgendes ändern:]
[1.Zeile: b durch d]
[2.Zeile: mit dd löschen und]
```

```
ls d*
a=100
print $a
datanal
diriname
100
$ fc -l -3 
78 history 72
79 history 69
80 ls d*
 a=100
 print $a
81 fc -l -3
$
```

[ mit p als letzte Zeile eintragen]  
[ neue 3.Zeile: HISTSIZE durch a ersetzen]  
[vi dann mit ZZ verlassen]  
:  
[Nach Verlassen des vi werden die entspr. geänderten 3 Kdos angezeigt]

[Ausgabe durch diese 3 Kommandos]

[die letzten 3 Kdos (+ aktuelles Kdo) anzeigen]

## 5.20 Die builtin-Editoren

Einer der größten Vorteile der **ksh** ist, daß sie das Editieren der momentanen oder zuvor eingegebener Kommandozeilen (aus der History-Datei) erlaubt. Dazu bietet sie die beiden builtin-Editoren **vi** und **emacs** an.

Anders als in der Bourne-Shell müssen bei einer fehlerhaften Kommandozeile nicht alle Zeichen »rückwärts« bis zu der Stelle, an der eine Änderung vorzunehmen ist, zunächst gelöscht und dann wieder neu eingegeben werden, sondern kann die Kommandozeile mit Hilfe von Editor-Kommandos editiert werden. Dies erleichtert nicht nur das interaktive Arbeiten mit der **ksh**, sondern hilft auch viel Zeit sparen, zumal nicht nur das Editieren der momentanen Kommandozeile, sondern auch früher eingegebener Kommandozeilen möglich ist.

Die beiden builtin-Editoren der **ksh** sind zeilenorientiert. Dies bedeutet, daß bis auf wenige Ausnahmen immer nur eine Zeile zu einem Zeitpunkt editiert werden kann.

### Spaltenzahl für die Editoren einstellen

Die **ksh** verwendet die Variable **COLUMNS**<sup>1</sup>, um die Spaltenzahl des jeweiligen Bildschirms festzulegen. Die Variable **TERM** wird dagegen nicht von der **ksh** für die Einstellung der beiden builtin-Editoren verwendet.

---

1. Siehe Kapitel 5.8.2.

## Das "Ein-Zeilen-Window" der ksh

**ksh**-Kommandozeilen können bis zu 256 Zeichen<sup>1</sup> lang sein. Wenn jedoch die **ksh** eine Kommandozeile aus der History-Datei und nicht vom Terminal liest, dann gilt diese Einschränkung von 256 Zeichen nicht.

Die **ksh** verfügt über ein »Ein-Zeilen-Window«, durch welches der Benutzer die momentane **ksh**-Zeile sieht. Die Window-Breite ist dabei durch die Variable **COLUMNS**<sup>2</sup> festgelegt. Wenn die momentane Kommandozeile länger als die Window-Breite minus 2 ist<sup>3</sup>, so schiebt die **ksh** das Window entsprechend nach links oder rechts, so daß immer nur ein Ausschnitt der vollständigen Kommandozeile am Bildschirm sichtbar ist. Dieses Verschieben führt die **ksh** automatisch durch, wenn der Cursor an den entsprechenden Rand des Windows bewegt wird.

Die **ksh** benutzt die folgende Notation in der letzten Spalte der Zeile, um anzuzeigen, daß nur ein Teil der Kommandozeile im Window dargestellt ist:

| Notation | Bedeutung                                     |
|----------|-----------------------------------------------|
| >        | rechts ist noch mehr Text vorhanden           |
| <        | links ist noch mehr Text vorhanden            |
| *        | links und rechts ist noch mehr Text vorhanden |

Vorsicht: Die Window-Breite wird durch die Länge des Promptstrings beeinflusst. Besonders, wenn Escape-Sequenzen im Prompt vorkommen, kann daraus eine falsch berechnete Window-Breite resultieren. Dies kann vermieden werden, wenn nach jeder Escape-Sequenz im Promptstring ein Neuezeile- oder Return-Zeichen angegeben wird.

## Unerlaubte Eingaben

Die **ksh** reagiert auf unerlaubte Eingaben mit dem Klingelton.

## Ein- bzw. Ausschalten der builtin-Editoren

Zu einem bestimmten Zeitpunkt kann nur einer der beiden builtin-Editoren eingeschaltet sein: entweder **emacs** oder **vi**. Der gewünschte builtin-Editor muß vom Systemadministrator in die **ksh** »einkompiliert« worden sein. Abhängig vom jeweiligen System können beide Editoren, einer von beiden oder keiner »einkompiliert« sein.

- 
1. Auf manchen Systemen sind auch mehr Zeichen für eine Kommandozeile zugelassen.
  2. Voreinstellung ist 80 Zeichen.
  3. Z.B. länger als 78 Zeichen, wenn **COLUMNS=80** ist.

### Einschalten während einer ksh-Sitzung

Es existieren drei verschiedene Möglichkeiten, während einer Sitzung einen Editor einzuschalten:

```
set -o vi bzw. set -o emacs
VISUAL=../vi bzw. VISUAL=../emacs
EDITOR=../vi bzw. EDITOR=../emacs
```

Bei **VISUAL** und **EDITOR** sind die absoluten Pfadnamen der entsprechenden Editoren anzugeben. Falls die beiden Variablen **VISUAL** und **EDITOR** gleichzeitig gesetzt sind, so hat **VISUAL** die höhere Priorität.

### Automatisches Einschalten bei jedem Anmelden

Eine der obigen Zeilen muß in der Datei *.profile* oder in der Environment-Datei angegeben werden.

### Ausschalten der builtin-Editoren

#### **set +o vi**

schaltet den builtin-Editor **vi** aus; danach ist kein builtin-Editor mehr eingeschaltet.

#### **set -o emacs**

schaltet den builtin-Editor **vi** aus und den builtin-Editor **emacs** ein.

### Unterschiede zwischen den beiden builtin-Editoren

- vi** editiert die Kommandozeilen der History-Datei. Da sich Kommandozeilen aus mehr als einer Zeile zusammensetzen können, ist also das Editieren eines mehrzeiligen Kommandos möglich.
- emacs** editiert die Zeilen der History-Datei. Dies bedeutet, daß immer nur eine Zeile zu einem bestimmten Zeitpunkt editiert werden kann. Bei einem mehrzeiligen Kommando müssen also die einzelnen Zeilen anders als im **vi** nacheinander editiert werden.

## 5.20.1 Builtin-Editor emacs

Mit **emacs** können sowohl die momentane Kommandozeile als auch Kommandozeilen aus der History-Datei editiert werden.

**emacs** wird in zwei Versionen angeboten: **emacs** und **gmacs**. Der einzige Unterschied zwischen diesen beiden Versionen ist, daß bei *Strg-t* folgendes gilt:

- emacs** vertauscht das Zeichen an der Cursorposition mit dem nächsten Zeichen.
- gmacs** vertauscht die beiden Zeichen links von der Cursorposition.

Der builtin-Editor **emacs** ist eine etwas geänderte Untermenge des wirklichen Editors **emacs** und verfügt nicht über alle **emacs**-Kommandos. Jedoch bietet der builtin-Editor **emacs** auch einige Zusätze an:

### Auflisten der Pfadnamen

**ESC=** listet die Pfadnamen auf, welche bei Expandierung des Worts (durch Anhängen von \*), auf dem der Cursor sich befindet, resultieren würden.

```
$ set -o emacs
$ ls bESC=
1) basename
2) bname
$ ls basename
basename
$ ls aESC=
1) a*
2) a.out
3) abc
4) abc2
5) add
6) addiere.c
7) ausgab
$ ls adESC=
1) add
2) addiere.c
$ ls ad*
add
addiere.c
$
```

### Vervollständigen eines Pfadnamens

**ESC ESC** hängt die fehlenden Zeichen an das Wort unter dem Cursor an, um es zu einem Pfadnamen einer existierenden Datei zu vervollständigen.

```
$ cd /usr/incESC ESC
 |
 V
$ cd /usr/include/
$ cd ~-
$ pwd
/user1/egon/kshueb
$
```

Angehängt werden dabei Zeichen bis zu einem Punkt, von dem ab mehrere Pfadnamen abgedeckt sind, oder bis der Pfadname vollständig ist. Wenn aus dieser Expandierung ein vollständiger Pfadname resultiert, so wird noch ein / angehängt, wenn es sich bei diesem Pfad-

namen um ein Directory handelt, ansonsten wird ein Leerzeichen angehängt. Diese Expandierungs-Möglichkeit mit *ESC ESC* ist jedoch nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurde. Auf Versionen vom 3.6.1986 muß anstelle von *ESC ESC* die Tastenfolge *ESC\** verwendet werden.

### Pfadnamen-Expandierung

**ESC\*** bewirkt, daß ein \* an das Wort unter dem Cursor angehängt wird, bevor dann Dateinamen-Expandierung für dieses Wort versucht wird. Wenn Expandierung für das Wort möglich ist, so wird das Wort durch den entsprechenden Dateinamen ersetzt, ansonsten bleibt das Wort unverändert, und es wird ein Klingelton ausgegeben.

```
$ print dESC*
 |
 V
$ print datanal diriname 
datanal diriname
$
```

### Einfügen des letzten Arguments

**ESC\_** fügt das letzte Argument der vorherigen Kommandozeile ein.

```
$ file ESC_
 |
 V
$ file diriname 
diriname: commands text
$
```

### Ausgabe der ksh-Versionsnummer

**Strg-v** gibt die Version der **ksh** aus.

### Abarbeiten von mehrzeiligen Kommandozeilen aus History-Datei

**Strg-o** übergibt die aktuelle Zeile an die **ksh** und holt die nächste Zeile aus der History-Datei. **Strg-o** wird verwendet, wenn mehrzeilige Kommandos aus der History-Datei auszuführen oder zu editieren sind.

Um eine Kommandozeile an die **ksh** zur Verarbeitung zu übergeben, ist die Taste  zu drücken. Viele der nachfolgenden **emacs**-Kommandos werden mit der Taste *ESC* eingeleitet. Diese Taste wird in **emacs**-Beschreibungen auch oft als *Meta-Taste* bezeichnet.

Folgende **emacs**-Direktiven stehen zur Verfügung:

### Cursorpositionierung

|                     |                                                                         |
|---------------------|-------------------------------------------------------------------------|
| <i>Strg-f</i>       | ( <b>f</b> orward) Cursor ein Zeichen nach rechts                       |
| <i>ESC nStrg-f</i>  | ( <b>f</b> orward) Cursor <i>n</i> Zeichen nach rechts                  |
| <i>Strg-b</i>       | ( <b>b</b> ackward) Cursor ein Zeichen nach links                       |
| <i>ESC nStrg-b</i>  | ( <b>b</b> ackward) Cursor <i>n</i> Zeichen nach links                  |
| <i>Strg-a</i>       | ( <b>a</b> nfang) Cursor an Zeilenanfang                                |
| <i>Strg-e</i>       | ( <b>e</b> nd) Cursor ans Zeilenende                                    |
| <i>Strg-]c</i>      | Cursor nach rechts auf das nächste Zeichen <i>c</i>                     |
| <i>ESC nStrg-]c</i> | Cursor nach rechts auf das <i>n</i> .te Vorkommen des Zeichens <i>c</i> |
| <i>ESCf</i>         | ( <b>f</b> orward) Cursor hinter das Cursor-Wort                        |
| <i>ESC nESCf</i>    | Cursor nach rechts hinter das <i>n</i> .te Wort                         |
| <i>ESCb</i>         | ( <b>b</b> ackward) Cursor auf Anfang des Cursor-Worts                  |
| <i>ESC nESCb</i>    | Cursor nach links auf Anfang des <i>n</i> .ten Worts                    |

In den nachfolgenden Beispielen wird die Cursorposition immer mit einem Unterstrich   angezeigt. Das entsprechende Editor-Kommando wird dabei immer links von der Kommandozeile angegeben, die aus diesem Editor-Kommando resultiert.

**Beispiel**

```

$ find / -name "*.c" -print_
ESCb $ find / -name "*.c" -print
Strg-a $ find / -name "*.c" -print
Esc3Escf $ find / -name "*.c " -print
Strg-b $ find / -name "*.c " -print
Strg-e $ find / -name "*.c" -print_
Esc3Escb $ find / -name "*.c" -print
Strg-]p $ find / -name "*.c" -print
Esc3Strg-f $ find / -name "*.c" -print
Strg-e $ find / -name "*.c" -print_

```

### Löschen

|                   |                                                                                                                                                                                                                          |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Erase</i>      | Zeichen links vom Cursor löschen. Die Voreinstellung für <i>Erase</i> ist meist <b>#</b> oder <i>Strg-h</i> oder die <i>Backspace</i> -Taste. Mit dem Kommando <b>stty</b> kann <i>Erase</i> auch anders gesetzt werden. |
| <i>ESC nErase</i> | <i>n</i> Zeichen links vom Cursor löschen                                                                                                                                                                                |

|                    |                                                                                                                                                                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Kill</i>        | ganze Zeilen löschen. Die Voreinstellung für <i>Kill</i> ist meist <b>@</b> oder <i>Strg-x</i> . Mit dem Kommando <b>stty</b> kann <i>Kill</i> auch anders gesetzt werden.                                                                                                                               |
| <i>Strg-k</i>      | <b>(kill)</b> Alle Zeichen ab Cursorposition bis zum Zeilenende löschen                                                                                                                                                                                                                                  |
| <i>ESC nStrg-k</i> | <b>(kill)</b> Wenn Cursor rechts von der Spalte <i>n</i> ist, so werden alle Zeichen von Spalte <i>n</i> bis zum Cursor (ohne Cursorzeichen) gelöscht.<br><br>Wenn Cursor links von der Spalte <i>n</i> ist, so werden alle Zeichen ab Cursorposition bis zur Spalte <i>n</i> (ausschließlich) gelöscht. |
| <i>Strg-d</i>      | <b>(delete)</b> Zeichen an Cursorposition löschen. Falls <i>Strg-d</i> das erste Zeichen einer Zeile ist, so interpretiert die <b>ksh</b> dies als <b>EOF</b> .                                                                                                                                          |
| <i>ESC nStrg-d</i> | <b>(delete)</b> <i>n</i> Zeichen ab Cursorposition (nach rechts) löschen                                                                                                                                                                                                                                 |
| <i>ESCd</i>        | <b>(delete)</b> ab Cursorposition bis zum Wortende löschen                                                                                                                                                                                                                                               |
| <i>ESC nESCd</i>   | <b>(delete)</b> ab Cursorposition (nach rechts) <i>n</i> Wörter löschen                                                                                                                                                                                                                                  |
| <i>ESC Strg-h</i>  |                                                                                                                                                                                                                                                                                                          |
| <i>ESC Strg-?</i>  |                                                                                                                                                                                                                                                                                                          |
| <i>ESCh</i>        | Ab Cursorposition (nach links) bis zum Wortanfang löschen. Anstelle von <i>Strg-h</i> kann auch die <i>Backspace</i> -Taste und anstelle von <i>Strg-?</i> die <i>Delete</i> -Taste eingegeben werden.                                                                                                   |
| <i>Strg-w</i>      | <b>(wipe out)</b> Ab Cursorposition bis zu einer markierten Stelle löschen. Eine Stelle kann mit <i>ESC Leertaste</i> markiert werden.                                                                                                                                                                   |

**Beispiel**

```

$ find / -name "*.c" -print_
ESC Strg-h $ find / -name "*.c" -_
Strg-a $ find / -name "*.c" -
Escd $ _/ -name "*.c" -
Esc5Strg-d $ ame "*.c" -
Strg-d $ me "*.c" -
Strg-k $ [Zeile nun leer]
.... [Neue Zeile eingeben]
$ find / -name -name -print "*.c" -print_
Esc3Escb $ find / -name -name -print "*.c" -print
Escf $ find / -name -name -print_ "*.c" -print
Esc Leertaste $ find / -name -name -print_ "*.c" -print
Esc2Escb $ find / -name -name -print "*.c" -print
Strg-b $ find / -name _name -print "*.c" -print
Strg-w $ find / -name _ "*.c" -print
Strg-h $ find / -name_ "*.c" -print
Strg-e $ find / -name "*.c" -print_

```



## Markieren und Kopieren

|                      |                                                                                                                                                                                                            |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ESC</i> Leertaste | Cursorposition markieren                                                                                                                                                                                   |
| <i>Strg-xStrg-x</i>  | ( <i>exchange</i> ) Cursor an markierte Stelle positionieren und Marke an ursprünglicher Cursorposition setzen. Mit dieser Direktive kann der Cursor an einer zuvor markierten Stelle positioniert werden. |
| <i>ESCp</i>          | ( <i>push</i> ) Text von Cursorposition bis markierte Stelle in einem Puffer zu sichern. Dieser Pufferinhalt kann dann an späterer Stelle mit <i>Strg-y</i> wieder einkopiert werden.                      |
| <i>Strg-y</i>        | ( <i>yank</i> ) Pufferinhalt an momentane Cursorposition kopieren                                                                                                                                          |

### Beispiel

```
$ find / -name "*.c" -print_
Strg-a $ find / -name "*.c" -print
ESCf $ find_ / -name "*.c" -print
ESC Leertaste $ find_ / -name "*.c" -print
Strg-e $ find / -name "*.c" -print_
Strg-x Strg-x $ find_ / -name "*.c" -print
Strg-x Strg-x $ find / -name "*.c" -print_
ESCp $ find / -name "*.c" -print_
Strg-y $ find / -name "*.c" -print / -name "*.c" -print_
ESC4ESCb $ find / -name "*.c" -print / -name "*.c" -print
ESCf $ find / -name "*.c" -print_ / -name "*.c" -print
Strg-k $ find / -name "*.c" -print_
```

## Sonstige emacs-Direktiven

|                    |                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Strg-t</i>      | ( <i>transpose</i> ) einzige Direktive, die in <b>emacs</b> und <b>gmacs</b> unterschiedlich wirkt:<br><br><b>emacs:</b> Zeichen an Cursorposition mit nachfolgendem Zeichen vertauschen und Cursor um eine Position nach rechts bewegen.<br><b>gmacs:</b> Beide Zeichen vor Cursorposition vertauschen; hier findet keine Cursor-Bewegung statt. |
| <i>Strg-c</i>      | ( <i>capitalize</i> ) Zeichen an Cursorposition durch entsprechenden Großbuchstaben ersetzen und Cursor ein Zeichen weiter positionieren.                                                                                                                                                                                                         |
| <i>ESC nStrg-c</i> | ( <i>capitalize</i> ) <i>n</i> Zeichen ab Cursorposition durch entsprechende Großbuchstaben ersetzen und Cursor <i>n</i> Zeichen weiter positionieren.                                                                                                                                                                                            |
| <i>ESCc</i>        | ( <i>capitalize</i> ) Ab Cursorposition bis Wortende alle Kleinbuchstaben durch Großbuchstaben ersetzen und Cursor auf Anfang des nächsten Words positionieren.                                                                                                                                                                                   |

|                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ESC nESCc</i>                                                                                                      | ( <i>capitalize</i> ) Ab Cursorposition bis zum Ende des <i>n</i> .ten Worts alle Kleinbuchstaben durch Großbuchstaben ersetzen und Cursor auf Anfang des <i>n</i> +1.ten Worts positionieren.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>ESCI</i>                                                                                                           | ( <i>lowercase</i> ) Ab Cursorposition bis Wortende alle Großbuchstaben durch Kleinbuchstaben ersetzen und Cursor auf Anfang des nächsten Worts positionieren.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>ESC nESCI</i>                                                                                                      | ( <i>lowercase</i> ) Ab Cursorposition bis zum Ende des <i>n</i> .ten Worts alle Großbuchstaben durch Kleinbuchstaben ersetzen und Cursor auf Anfang des <i>n</i> +1.ten Worts positionieren.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <i>Strg-l</i>                                                                                                         | ( <i>line redraw</i> ) Cursor in die nächste Zeile bewegen und momentane Kommandozeile erneut anzeigen; wird verwendet, wenn eine Kommandozeile, z.B. bedingt durch Steuerzeichen, nicht mehr den wirklichen Text beinhaltet, um sich diese wieder bereinigt anzeigen zu lassen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>EOF</i>                                                                                                            | <p>(<i>end of file</i>) wird nur erkannt, wenn es das erste Zeichen einer Zeile ist, ansonsten wird es als normales Zeichen (z.B. <b>emacs</b>-Direktive) interpretiert. Die Voreinstellung für <i>EOF</i> ist <i>Strg-d</i>; mit dem Kommando <b>stty</b> kann <i>EOF</i> jedoch anders gesetzt werden.</p> <p>Wenn die Option <b>ignoreeof</b><sup>a</sup> nicht gesetzt ist, so kann mit <i>EOF</i> die aktuelle <b>ksh</b> verlassen werden. Ist dagegen die Option <b>ignoreeof</b> gesetzt, so kann eine <b>ksh</b> nur mit der Eingabe des Kommandos <b>exit</b> verlassen werden.</p>                                                                                                                                                            |
| <i>Strg-j</i><br><i>Strg-m</i><br> | momentane Kommandozeile zur Ausführung an die <b>ksh</b> übergeben.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>ESC=</i>                                                                                                           | listet die Pfadnamen auf, welche aus der Expandierung des Cursor-Worts (durch Anhängen von *) resultieren würden.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <i>ESC ESC</i>                                                                                                        | <p>hängt die fehlenden Zeichen an das Wort unter dem Cursor an, um es zu einem Pfadnamen einer existierenden Datei zu vervollständigen. Angehängt werden dabei Zeichen bis zu einem Punkt, von dem ab mehrere Pfadnamen abgedeckt sind, oder bis der Pfadname vollständig ist. Wenn aus dieser Expandierung ein vollständiger Pfadname resultiert, so wird noch ein / angehängt, wenn es sich bei diesem Pfadnamen um ein Directory handelt, ansonsten wird ein Leerzeichen angehängt.</p> <p>Diese Expandierungsmöglichkeit mit <i>ESC ESC</i> ist jedoch nur auf <b>ksh</b>-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurde. Auf Versionen vom 3.6.1986 muß anstelle von <i>ESC ESC</i> die Tastenfolge <i>ESC*</i> verwendet werden.</p> |

|                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ESC*</i>                  | bewirkt, daß ein * an das Wort unter dem Cursor angehängt wird, bevor dann Dateinamen-Expandierung für dieses Wort versucht wird. Wenn Expandierung für das Wort möglich ist, so wird das Wort durch die entsprechenden Dateinamen ersetzt, ansonsten bleibt das Wort unverändert, und es wird ein Klingelton ausgegeben.                                                                                                                                                                                      |
| <i>Strg-u</i>                | nachfolgende <b>emacs</b> -Direktive 4mal ausführen.                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>\</i>                     | Wirkung der nachfolgenden <b>emacs</b> -Direktive ausschalten und diese als normales Zeichen an Cursorposition einfügen.                                                                                                                                                                                                                                                                                                                                                                                       |
| <i>Strg-v</i>                | ( <b>version</b> ) Datum und Kurzbezeichnung der <b>ksh</b> -Version ausgeben.<br><br>Kommandoeingabe kann nach Drücken einer beliebigen Taste fortgesetzt werden.                                                                                                                                                                                                                                                                                                                                             |
| <i>ESC buchstabe</i>         | durchsucht die Alias-Liste nach einem Alias-Namen <i>_buchstabe</i> . Falls ein solcher Alias-Name existiert, wird dessen Wert von der <b>ksh</b> an der Cursorposition eingefügt bzw. die darin enthaltenen <b>emacs</b> -Direktiven werden ausgeführt.<br><br><i>buchstabe</i> darf keiner der von <b>emacs</b> nach <i>ESC</i> benutzten Buchstaben ( <b>f,b,d,p,l,c,h</b> ) sein; auch sollten nur Großbuchstaben benutzt werden, um mögliche Konflikte in zukünftigen <b>ksh</b> -Versionen zu vermeiden. |
| <i>ESC .</i><br><i>ESC _</i> | an Cursorposition letztes Wort der vorherigen Kommandozeile einfügen.                                                                                                                                                                                                                                                                                                                                                                                                                                          |

a. Siehe Kapitel 5.24.3.

**Beispiel**

```
$ set -o gmacs 
$ print_
ESC3Strg-b $ print
Strg-t $ rpint
Strg-t $ print
Strg-a $ p_rint
Strg-k $ _

$ set -o emacs 
$ print_
ESC3Strg-b $ print
Strg-t $ prnit
Strg-t $ prnti
Strg-a $ prnti
Strg-c $ P_rnti
ESC3Strg-c $ PRNTi
Strg-a $ _PRNTi
Strg-k $ _
```

```
$ print Hallo wie geht es emil_
Strg-a $ print Hallo wie geht es emil
ESCf $ print_Hallo wie geht es emil
ESC3ESCc $ print HALLO WIE GEHT_es emil
ESC2ESCb $ print HALLO WIE GEHT es emil
ESC3ESC1 $ print HALLO wie geht es_emil
Strg-uStrg-b $ print HALLO wie geh_t es emil1
Strg-uESCb $ print HALLO wie geht es emil
Strg-uESCf $ print HALLO wie geht_es emil
\Strg-a $ print HALLO wie geht^A_es emil
Strg-a $ print HALLO wie geht^A es emil
Strg-k $

$ alias _K='\ESC Leertaste\Strg-a\ESCf\Strg-f"\Strg-e"\Strg-x\Strg-x' 
$ print Guten Morgen,\n lieber Emil_
Escb $ print Guten Morgen,\n lieber _emil
EscK $ print "Guten Morgen,\n lieber_Emil"

Strg-m Ausgabe: Guten Morgen,
 lieber Emil
$
```

## Kommandozeilen aus der History-Datei holen

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Strg-p</i>      | <p>(previous) vorherige Kommandozeile aus der History-Datei holen. Eine erneute Eingabe von <i>Strg-p</i> holt dann die Kommandozeile vor der vorherigen Kommandozeile usw.</p> <p>Wenn sowohl <b>emacs</b> als auch <b>vi</b> benutzt werden, muß folgendes beachtet werden:</p> <p><b>emacs</b> bearbeitet nur einzelne Zeilen aus der History-Datei.</p> <p><b>vi</b> dagegen bearbeitet ganze Kommandozeilen (eventuell mehrere Zeilen) aus der History-Datei.</p> <p>Um in <b>emacs</b> Kommandozeilen aus der History-Datei zu bearbeiten, die aus mehr als einer Zeile bestehen, muß <i>Strg-o</i> (nicht ) verwendet werden. <i>Strg-o</i> übergibt die entsprechende Zeile an die <b>ksh</b> zur Bearbeitung und holt zugleich die nachfolgende Zeile aus der History-Datei.</p> |
| <i>ESC nStrg-p</i> | (previous) <i>n</i> .te vorherige Kommandozeile aus der History-Datei holen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>ESC&lt;</i>     | älteste (erste) Zeile aus der History-Datei holen <sup>a</sup> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <i>ESC&gt;</i>     | letzte Zeile aus der History-Datei holen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

1. Auf manchen Systemen steht *Strg-u* für *Kill* (Zeile löschen). Soll dort diese emacs-Direktive verwendet werden, müßte mit **stty Kill** umgesetzt werden;  
z.B. **stty kill ^n**

|                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Strg-n</b>                                                                                               | ( <b>next</b> ) auf die nächste Zeile in der History-Datei weiterschalten; dies bedeutet: die nächste Zeile nach der zuletzt geholten Zeile aus der History-Datei holen.                                                                                                                                                                                                                                                           |
| <b>ESC nStrg-n</b>                                                                                          | ( <b>next</b> ) Um <i>n</i> Zeilen in der History-Datei weiterschalten und dann diese holen.                                                                                                                                                                                                                                                                                                                                       |
| <b>Strg-r[string]</b>      |                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>ESC0Strg-r[string]</b>  | durchsucht die History-Datei nach dem ersten Vorkommen einer Kommandozeile, die <i>string</i> enthält. Wenn angegeben ist:<br><br><i>string</i> und <b>ESC0</b> : Vorwärts suchen<br>nur <i>string</i> : Rückwärts suchen<br>nur <b>ESC0</b> : Rückwärts suchen ab der Position der vorherigen Suche<br><br>weder <i>string</i> noch <b>ESC0</b> : nächste Kommandozeile holen, die den zuletzt angegebenen <i>string</i> enthält. |
| <b>Strg-r^string</b>       |                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>ESC0Strg-r^string</b>   | wie Direktive zuvor, mit dem Unterschied, daß der angegebene <i>string</i> nur dann eine Zeile abdeckt, wenn dieser am Zeilenanfang steht.<br><br>Diese Direktive ist nur auf <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.                                                                                                                                                                           |
| <b>Strg-o</b>                                                                                               | ( <b>operate</b> ) übergibt die aktuelle Zeile an die <b>ksh</b> zur Verarbeitung und holt die nächste Zeile aus der History-Datei; wird verwendet, um Kommandos ausführen zu lassen, die sich über mehrere Zeilen erstrecken.                                                                                                                                                                                                     |

- a. Es ist zu beachten, daß in der History-Datei immer nur so viele Kommandozeilen aufgeho-  
ben werden, wie mit der Variablen **HISTSIZE** festgelegt ist.

**Beispiel**

```
$ PS1="! $ " 
160 $ date 
Tue Jun 11 09:13:23 MESZ 1996
161 $ print $OLDPWD 
/user1/egon
162 $ alias type="cat " 
163 $ for i in b* 
> do wc -l $i 
> done 
 1 basename
 10 bname
164 $ find $HOME -name isnum -print 
/user1/egon/flib/isnum
165 $ history -5 
160 date
161 print $OLDPWD
```

```

162 alias type="cat "
163 for i in b*
164 do wc -l $i
165 done
166 find $HOME -name isnum -print
167 history -5
168 $
ESC> 166 $ history -5_
Strg-p 166 $ find $HOME -name isnum -print_
Strg-p 166 $ for i in b*_
Strg-o > do wc -l $i_
Strg-o > done_
Strg-m 1 basisname
 10 bname
 167 $
Strg-p 167 $ for i in b*_
Strg-rcat ↵ 167 $ alias type="cat " _
Strg-rda ↵ 167 $ date_
Strg-n 167 $ print $OLDPWD_
ESC0Strg-r^fin ↵ 167 $ find $HOME -name isnum -print_
ESC2ESCb 167 $ find $HOME -name isnum -print
ESC5Strg-d 167 $ find $HOME -name _-print
add 167 $ find $HOME -name add_-print
↵ /user1/egon/kshueb/add
 168 $

```

## 5.20.2 Builtin-Editor vi

Der builtin-Editor **vi** kennt zwei Arbeitszustände

- ▶ Eingabemodus
- ▶ Kommandomodus

Anders als der wirkliche **vi** befindet sich der builtin-Editor immer zu Beginn im Eingabemodus. Vom Eingabemodus in den Kommandomodus kann mit **ESC** umgeschaltet werden.

Der builtin-Editor **vi** ist eine leicht abgeänderte Untermenge des wirklichen Editors **vi** und verfügt somit nicht über alle **vi**-Kommandos. Gleiche Kommandonamen haben manchmal sogar unterschiedliche Wirkungen.

## Zusätze gegenüber dem wirklichen vi

Der builtin-Editor **vi** bietet folgende zusätzliche Funktionalität:

### Kommentar

# trägt die momentane Kommandozeile als Kommentar in die History-Datei ein.

### Auflisten von Pfadnamen

= (Gleichheitszeichen) listet die Pfadnamen auf, welche aus der Expandierung (bei Anhängen von \*) des Cursor-Wortes resultieren würden.

```
$ set -o vi 
$ ls bESC=
1) basisname
2) bname
$ ls baaisname  [erstes eingegeb. a für Umschalten in Eingabemodus
 (append)]
basisname
$ ls aESC=
1) a*
2) a.out
3) abc
4) abc2
5) add
6) addiere.c
7) ausgab
$ ls aadESC= [erstes eingegebene a für Umschalten in Eingabemodus
 (append)]
1) add
2) addiere.c
$ ls ada*  [erstes eingegebene a für Umschalten in Eingabemodus
 (append)]
add
addiere.c
$
```

### Vervollständigen eines Pfadnamens

\ hängt die fehlenden Zeichen an das Wort unter dem Cursor an, um es zu einem Pfadnamen einer existierenden Datei zu vervollständigen. Angehängt werden dabei Zeichen bis zu einem Punkt, von dem ab mehrere Pfadnamen abgedeckt sind, oder bis der Pfadname vollständig ist. Wenn aus dieser Expandierung ein vollständiger Pfadname resultiert, so wird noch ein / angehängt, wenn es sich bei diesem Pfadnamen um ein Directory handelt, ansonsten wird ein Leerzeichen angehängt.

Diese Direktive \ ist jedoch nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.

```
$ cd /usr/incESC\
 |
 V
$ cd /usr/include/ ↵
$ cd ~- ↵
$ pwd ↵
/user1/egon/kshueb
$
```

### Pfadnamen-Expandierung

\* bewirkt, daß ein \* an das Wort unter dem Cursor angehängt wird, bevor dann Dateinamen-Expandierung für dieses Wort versucht wird. Wenn eine Expandierung für das Wort möglich ist, so wird das Wort durch den entsprechenden Dateinamen ersetzt, ansonsten bleibt das Wort unverändert, und es wird ein Klingelton ausgegeben.

```
$ print dESC*
 |
 V
$ print datanal diriname ↵
datanal diriname
$
```

### Aufruf des wirklichen vi

**v** ruft den wirklichen Editor **vi** auf. In der zu editierenden Datei befindet sich dabei die aktuelle Kommandozeile, die nun mit dem wirklichen **vi** editiert werden kann. Nach dem Verlassen von **vi** wird diese Kommandozeile dann von der **ksh** ausgeführt.

```
$ find / -name eingabe -printESCv
.....
[vi wird mit dieser Zeile als Inhalt aufgerufen]
[Nach Ändern dieser Zeile im vi zu:]
[find $HOME -name eingabe -print]
[und Verlassen des vi mit ZZ, wird diese Kommandozeile ausgeführt]
find $HOME -name eingabe -print
/user1/egon/kshueb/eingabe
$
```



Nachfolgend wird folgende Notation benutzt:

| Schreibweise   | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>vi-WORT</i> | ist eine Folge von Buchstaben, Ziffern und / oder Interpunktionszeichen, welche durch Neuezeile-Zeichen, Tabulatorzeichen oder Leerzeichen abgegrenzt ist. Interpunktionszeichen gelten nur dann als eigene <i>vi-WORTE</i> , wenn um sie herum (auf beiden Seiten) Leer- oder Tabulatorzeichen angegeben sind. So zählt z.B. eine zusammenhängende Folge von Interpunktionszeichen als ein <i>vi-WORT</i> . |
| <i>vi-wort</i> | ist eine Folge von Buchstaben und / oder Ziffern, die durch Neuezeile-Zeichen, Leerzeichen, Tabulatorzeichen oder einem Interpunktionszeichen abgegrenzt ist.                                                                                                                                                                                                                                                |

## Eingabemodus

Im Eingabemodus stehen neben der normalen Text-Eingabe folgende Editiermöglichkeiten zur Verfügung:

| Kommando      | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Erase</i>  | Zeichen links vom Cursor löschen. Die Voreinstellung für <i>Erase</i> ist meist <b>#</b> oder <i>Strg-h</i> oder die <i>Backspace</i> -Taste. Mit dem Kommando <b>stty</b> kann <i>Erase</i> auch anders gesetzt werden.                                                                                                                                                                                                                                                                                                                                   |
| <i>Kill</i>   | ganze Zeilen löschen. Die Voreinstellung für <i>Kill</i> ist meist <b>@</b> oder <i>Strg-x</i> . Mit dem Kommando <b>stty</b> kann <i>Kill</i> auch anders gesetzt werden.                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>Strg-v</i> | nachfolgendes <b>vi</b> -Kommando ausschalten und als einfachen Text interpretieren.                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>\</b>      | ähnlich zu <i>Strg-v</i> , jedoch kann mit <b>\</b> nur ein nachfolgendes <i>Erase</i> oder <i>Kill</i> ausgeschaltet werden.                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <i>EOF</i>    | <b>(end of file)</b> wird nur erkannt, wenn es das erste Zeichen einer Zeile ist, ansonsten wird es als normales Zeichen interpretiert. Die Voreinstellung für <i>EOF</i> ist <i>Strg-d</i> ; mit dem Kommando <b>stty</b> kann <i>EOF</i> jedoch anders gesetzt werden.<br><br>Wenn die Option <b>ignoreeof</b> <sup>a</sup> nicht gesetzt ist, so kann mit <i>EOF</i> die aktuelle <b>ksh</b> verlassen werden. Ist dagegen die Option <b>ignoreeof</b> gesetzt, so kann eine <b>ksh</b> nur mit der Eingabe des Kommandos <b>exit</b> verlassen werden. |
| <i>Strg-w</i> | löscht das vorherige <i>vi-wort</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

a. Siehe Kapitel 5.24.3.

Die nachfolgenden **vi**-Kommandos können nur im Kommandomodus eingegeben werden. Mit **ESC** kann vom Eingabemodus in den Kommandomodus umgeschaltet werden.

| <i>Cursorpositionierung (Kommandomodus)</i> |                                                                                                                                                                                                                                          |
|---------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>I</b><br><i>Leerzeichen</i>              | Cursor ein Zeichen nach rechts                                                                                                                                                                                                           |
| <b>nI</b><br><i>nLeerzeichen</i>            | Cursor <i>n</i> Zeichen nach rechts                                                                                                                                                                                                      |
| <b>w</b>                                    | ( <b>w</b> ord) Cursor nach rechts auf Anfang des nächsten <i>vi-worts</i>                                                                                                                                                               |
| <b>nw</b>                                   | ( <b>w</b> ord) Cursor nach rechts auf Anfang des <i>n</i> .ten nächsten <i>vi-worts</i>                                                                                                                                                 |
| <b>W</b>                                    | ( <b>W</b> ord) Cursor nach rechts auf Anfang des nächsten <i>vi-WORTS</i>                                                                                                                                                               |
| <b>nW</b>                                   | ( <b>W</b> ord) Cursor nach rechts auf Anfang des <i>n</i> .ten nächsten <i>vi-WORTS</i>                                                                                                                                                 |
| <b>e</b>                                    | ( <b>e</b> nd) Cursor nach rechts auf Ende des nächsten <i>vi-worts</i>                                                                                                                                                                  |
| <b>ne</b>                                   | ( <b>e</b> nd) Cursor nach rechts auf Ende des <i>n</i> .ten nächsten <i>vi-worts</i>                                                                                                                                                    |
| <b>E</b>                                    | ( <b>E</b> nd) Cursor nach rechts auf Ende des nächsten <i>vi-WORTS</i>                                                                                                                                                                  |
| <b>nE</b>                                   | ( <b>E</b> nd) Cursor nach rechts auf Ende des <i>n</i> .ten nächsten <i>vi-WORTS</i>                                                                                                                                                    |
| <b>h</b>                                    | Cursor ein Zeichen nach links                                                                                                                                                                                                            |
| <b>nh</b>                                   | Cursor <i>n</i> Zeichen nach links                                                                                                                                                                                                       |
| <b>b</b>                                    | ( <b>b</b> ack) Cursor auf Anfang des <i>vi-worts</i> (links vom Cursor)                                                                                                                                                                 |
| <b>nb</b>                                   | ( <b>b</b> ack) Cursor auf Anfang des <i>n</i> .ten <i>vi-worts</i> (links vom Cursor)                                                                                                                                                   |
| <b>B</b>                                    | ( <b>B</b> ack) Cursor auf Anfang des <i>vi-WORTS</i> (links vom Cursor)                                                                                                                                                                 |
| <b>nB</b>                                   | ( <b>B</b> ack) Cursor auf Anfang des <i>n</i> .ten <i>vi-WORTS</i> (links vom Cursor)                                                                                                                                                   |
| <b>^</b>                                    | Cursor auf das erste Zeichen der Zeile, das kein Leer- oder Tabulatorzeichen ist.                                                                                                                                                        |
| <b>0</b>                                    | Cursor auf das erste Zeichen der Zeile                                                                                                                                                                                                   |
| <b>\$</b>                                   | Cursor auf das letzte Zeichen der Zeile                                                                                                                                                                                                  |
| <b> </b>                                    | Cursor auf das erste Zeichen der Zeile                                                                                                                                                                                                   |
| <b>n </b>                                   | Cursor auf das <i>n</i> .te Zeichen der Zeile.<br><br>Falls <i>n</i> größer als die Zeilenlänge ist, so wird Cursor auf das letzte Zeichen positioniert.<br><br>Nur auf nach dem 3.6.1986 freigegebenen <b>ksh</b> -Versionen verfügbar. |
| <b>fc</b>                                   | ( <b>f</b> ind) Cursor nach rechts auf das nächste Vorkommen des Zeichens <i>c</i>                                                                                                                                                       |
| <b>nfc</b>                                  | ( <b>f</b> ind) Cursor nach rechts auf das <i>n</i> .te Vorkommen des Zeichens <i>c</i>                                                                                                                                                  |
| <b>Fc</b>                                   | ( <b>F</b> ind) Cursor nach links auf das nächste Vorkommen des Zeichens <i>c</i>                                                                                                                                                        |
| <b>nFc</b>                                  | ( <b>F</b> ind) Cursor nach links auf das <i>n</i> .te Vorkommen des Zeichens <i>c</i>                                                                                                                                                   |

| <i>Cursorpositionierung (Kommandomodus)</i> |                                                                                                                                                                                                                    |
|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>tc</b>                                   | ( <b>to</b> ) Cursor nach rechts vor das nächste Vorkommen des Zeichens <i>c</i>                                                                                                                                   |
| <b>ntc</b>                                  | ( <b>to</b> ) Cursor nach rechts vor das <i>n</i> .te Vorkommen des Zeichens <i>c</i>                                                                                                                              |
| <b>Tc</b>                                   | ( <b>Tb</b> ) Cursor nach links nach dem nächsten Vorkommen des Zeichens <i>c</i>                                                                                                                                  |
| <b>nTc</b>                                  | ( <b>Tb</b> ) Cursor nach links nach dem <i>n</i> .ten Vorkommen des Zeichens <i>c</i>                                                                                                                             |
| <b>;</b>                                    | wiederholt das zuletzt gegebene Kommando <b>f</b> , <b>F</b> , <b>t</b> oder <b>T</b>                                                                                                                              |
| <b>n;</b>                                   | wiederholt das zuletzt gegebene Kommando <b>f</b> , <b>F</b> , <b>t</b> oder <b>T</b> <i>n</i> -mal.<br><br>; kann auch mehrmals nacheinander angegeben werden; so entspricht z.B. <b>;;;</b> der Angabe <b>3;</b> |
| <b>,</b>                                    | wiederholt das zuletzt gegebene Kommando <b>f</b> , <b>F</b> , <b>t</b> oder <b>T</b> in umgekehrter Richtung.                                                                                                     |
| <b>n,</b>                                   | wiederholt das zuletzt gegebene Kommando <b>f</b> , <b>F</b> , <b>t</b> oder <b>T</b> <i>n</i> -mal in umgekehrter Richtung.<br><br>Dieses Kommando ist nützlich, wenn versehentlich zu weit positioniert wurde.   |

In den nachfolgenden Beispielen wird die Cursor-Position immer mit einem Unterstrich   angezeigt. Das entsprechende Editor-Kommando wird dabei immer links von der Kommandozeile angegeben, die aus diesem Editor-Kommando resultiert.

**Beispiel**

```

$ find / -name "*.c" -print_
ESCb $ find / -name "*.c" -print
0 $ find / -name "*.c" -print
4E $ find / -name "*.c " -print
h $ find / -name "*.c " -print
$ $ find / -name "*.c" -print
b $ find / -name "*.c" -print
6| $ find L -name "*.c" -print
fm $ find / -name "*.c" -print
Fi $ find / -name "*.c" -print
tr $ find / -name "*.c" -print
Ta $ find / -name "*.c" -print
0 $ find / -name "*.c" -print
W $ find L -name "*.c" -print
W $ find / name "*.c" -print
W $ find / -name " *.c" -print
W $ find / -name " *.c" -print
0 $ find / -name "*.c" -print
w $ find L -name "*.c" -print
w $ find / name " *.c" -print
w $ find / -name " *.c" -print
w $ find / -name " *.c" -print

```

```
w $ find / -name "*.c" -print
w $ find / -name "*.c_" -print
w $ find / -name "*.c_"_print
w $ find / -name "*.c" -print
w $ find / -name "*.c" -print_
0 $ find / -name "*.c" -print
dd $ _
```

## Einfügen, Ändern und Ersetzen (Kommandomodus)

Bis auf das Kommando **r** (*replace*) schalten die nachfolgenden Kommandos den builtin-Editor **vi** in den Eingabemodus um. Im Eingabemodus kann dann beliebiger Text eingegeben werden.

Für die Rückkehr vom Eingabemodus in den Kommandomodus, muß *ESC* eingegeben werden. Eine andere Möglichkeit ist die Eingabe von ; in diesem Fall wird die Kommandozeile sofort zur Ausführung an die **ksh** übergeben.

| Kommando                   | Wirkung                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>a</b>                   | ( <i>append</i> ) rechts von Cursorposition Text einfügen                                                                                                                                                                                                                                                                                                                                                                     |
| <b>A</b>                   | ( <i>Append</i> ) am Zeilenende Text einfügen; identisch zu <b>\$a</b>                                                                                                                                                                                                                                                                                                                                                        |
| <b>i</b>                   | ( <i>insert</i> ) links von Cursorposition Text einfügen                                                                                                                                                                                                                                                                                                                                                                      |
| <b>I</b>                   | ( <i>Insert</i> ) am Zeilenanfang (vor dem ersten Zeichen, das kein Leer- oder Tabulatorzeichen ist) Text einfügen; identisch zu <b>^i</b>                                                                                                                                                                                                                                                                                    |
| <b>R</b>                   | ( <i>Replace</i> ) auf Überschreiben (im Eingabemodus) schalten: Jedes danach eingegebene Zeichen ersetzt das Zeichen an der Cursorposition, bevor der Cursor weiter positioniert wird.                                                                                                                                                                                                                                       |
| <b>cp</b>                  | ( <i>change</i> ) <i>p</i> muß ein Cursorpositionierungs-Kommando sein. Es legt einen Bereich fest, der sich von der Cursorposition bis zu der mit <i>p</i> gewählten Position erstreckt.<br><br>Der nachfolgend eingegebene Text ersetzt dann den Text in diesem Bereich. Anders als im wirklichen <b>vi</b> wird jedoch der Text aus dem festgelegten Bereich zuerst gelöscht, bevor in den Eingabemodus umgeschaltet wird. |
| <b>cn p</b><br><b>n cp</b> | ( <i>change</i> ) legt <i>n</i> Bereiche zum Ändern fest                                                                                                                                                                                                                                                                                                                                                                      |
| <b>cc</b>                  | die ganze Eingabezeile löschen und in den Eingabemodus umschalten                                                                                                                                                                                                                                                                                                                                                             |
| <b>C</b>                   | ( <i>Change</i> ) ab Cursorposition Rest der Zeile löschen und in Eingabemodus umschalten; identisch zu <b>c\$</b>                                                                                                                                                                                                                                                                                                            |
| <b>S</b>                   | ( <i>Substitute</i> ) ganze Zeile löschen und in Eingabemodus umschalten; identisch zu <b>cc</b>                                                                                                                                                                                                                                                                                                                              |
| <b>rc</b>                  | ( <i>replace</i> ) ersetzt das Zeichen an Cursorposition durch das Zeichen <i>c</i> ; hier wird nicht in den Eingabemodus umgeschaltet                                                                                                                                                                                                                                                                                        |

| Kommando   | Wirkung                                                                                                                                                                                                                                                                                                           |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>nrc</i> | ( <i>replace</i> ) ab Cursorposition <i>n</i> Zeichen durch das Zeichen <i>c</i> ersetzen; nach der Ersetzung wird Cursor auf das zuletzt geänderte Zeichen positioniert und man befindet sich wieder im Kommandomodus.<br><br>Ist nur auf <b>ksh</b> -Versionen verfügbar, die nach 3.6.1986 freigegeben wurden. |
| <i>_</i>   | (Unterstrich) Letztes Wort der vorherigen Kommandozeile nach Cursorposition einfügen und in den Eingabemodus umschalten.                                                                                                                                                                                          |
| <i>n_</i>  | (Unterstrich) <i>n</i> .tes <i>vi</i> -WORT der vorherigen Kommandozeile (von Beginn an) nach Cursorposition einfügen und in den Eingabemodus umschalten                                                                                                                                                          |

**Beispiel**

```
$ print Hallo emil_
ESC $ print Hallo emil_
2b $ print Hallo emil
cwServusESC $ print Servus_ emil
a,ESC $ print Servus_ emil
ll $ print Servus, _emil
cILieber EESC $ print Servus, Lieber _Emil
7h $ print Servus, _Lieber Emil
cfESehr Herr EESC $ print Servus, Sehr Herr _Emil
4b $ print _Servus, Sehr Herr Emil
cwGuten TagESC $ print Guten Tag, _Sehr Herr Emil
W $ print Guten Tag, _Sehr Herr Emil
rs $ print Guten Tag, _sehr Herr Emil
w $ print Guten Tag, sehr _Herr Emil
igeehrter ESC $ print Guten Tag, sehr geehrter_Herr Emil
0 $ print Guten Tag, sehr geehrter Herr Emil
w $ print Guten Tag, sehr geehrter Herr Emil
i"ESC $ print _Guten Tag, sehr geehrter Herr Emil
A"ESC $ print "Guten Tag, sehr geehrter Herr Emil"
0 $ print "Guten Tag, sehr geehrter Herr Emil"
cc $ _
```

## Löschen (Kommandomodus)

| Kommando  | Wirkung                                           |
|-----------|---------------------------------------------------|
| <b>x</b>  | Zeichen an Cursorposition löschen                 |
| <b>nx</b> | <i>n</i> Zeichen ab Cursorposition löschen        |
| <b>X</b>  | Zeichen links von Cursorposition löschen          |
| <b>nX</b> | <i>n</i> Zeichen links von Cursorposition löschen |

| Kommando                  | Wirkung                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>dp</b>                 | ( <b>delete</b> ) <i>p</i> muß ein Cursorpositionierungs-Kommando sein. Es legt einen Bereich fest, der sich von der Cursorposition bis zu der mit <i>p</i> gewählten Position erstreckt.<br><br>Der so festgelegte Bereich wird gelöscht, wobei der gelöschte Text in einem Puffer gesichert wird. Der Inhalt dieses Puffers kann mit <b>u</b> oder <b>p</b> wieder einkopiert werden. |
| <b>dn p</b><br><b>ndp</b> | ( <b>delete</b> ) löscht <i>n</i> Bereiche                                                                                                                                                                                                                                                                                                                                              |
| <b>dd</b>                 | ( <b>delete</b> ) ganze Zeile löschen                                                                                                                                                                                                                                                                                                                                                   |
| <b>D</b>                  | ( <b>Delete</b> ) ab Cursorposition den Rest der Zeile löschen; identisch zu <b>d\$</b>                                                                                                                                                                                                                                                                                                 |

**Beispiel**

```
$ print Hallo emil, wie gehts denn so_
ESC $ print Hallo emil, wie gehts denn so
7b $ print Hallo emil, wie gehts denn so
x $ print allo emil, wie gehts denn so
3x $ print o emil, wie gehts denn so
df, $ print _wie gehts denn so
dw $ print wie gehts denn so
2dw $ print denn so
D $ print_
dd $ _
```

## Kopieren (Kommandomodus)

| Kommando                  | Wirkung                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>yp</b>                 | ( <b>yank</b> ) <i>p</i> muß ein Cursorpositionierungs-Kommando sein. Es legt einen Bereich fest, der sich von der Cursorposition bis zu der mit <i>p</i> gewählten Position erstreckt.<br><br>Der so festgelegte Bereich wird in einen Puffer kopiert <sup>a</sup> .<br><br>Der Inhalt dieses Puffers kann später mit einem der Kommandos <b>p</b> oder <b>P</b> wieder einkopiert werden.<br><br>Der Text und die Cursorposition werden durch <b>y</b> nicht verändert. |
| <b>yn p</b><br><b>nyp</b> | ( <b>yank</b> ) kopiert <i>n</i> Bereiche in den Puffer                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>yy</b>                 | ( <b>yank</b> ) ganze Zeile in Puffer kopieren                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>Y</b>                  | ( <b>Yank</b> ) ab Cursorposition den Rest der Zeile in den Puffer kopieren <sup>b</sup> ; identisch zu <b>y\$</b>                                                                                                                                                                                                                                                                                                                                                        |
| <b>p</b>                  | ( <b>put</b> ) den Pufferinhalt rechts vom Cursor einkopieren                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>np</b>                 | ( <b>put</b> ) den Pufferinhalt rechts vom Cursor <i>n</i> -mal einkopieren                                                                                                                                                                                                                                                                                                                                                                                               |

| Kommando  | Wirkung                                                                    |
|-----------|----------------------------------------------------------------------------|
| <b>P</b>  | ( <b>Put</b> ) den Pufferinhalt links vom Cursor einkopieren               |
| <b>nP</b> | ( <b>Put</b> ) den Pufferinhalt links vom Cursor <i>n</i> -mal einkopieren |

- a. Vorheriger Inhalt dieses Puffers wird überschrieben.
- b. Anders als im wirklichen **vi**.

**Beispiel**

```
$ print ----- Titel_
ESC $ print ----- Titel_
2b $ print _----- Titel
yw $ print _----- Titel
$ $ print ----- Titel_
p $ print ----- Titel-----_
0 $ _print ----- Titel-----
w $ print _----- Titel-----
y3l $ print _----- Titel-----
w $ print ----- _itel-----
l $ print ----- Titel-----
P $ print ----- T--itel-----
0 $ _print ----- T---itel-----
dd $ _
```

**Rückgängig machen (Kommandomodus)**

| Kommando | Wirkung                                                                                                                                                                                                                                               |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>u</b> | ( <b>undo</b> ) die letzte durch ein Editor-Kommando bewirkte Text-Änderung rückgängig machen                                                                                                                                                         |
| <b>U</b> | ( <b>Undo line</b> ) alle durch Editor-Kommandos vorgenommenen Text-Änderungen in der momentanen Zeile rückgängig machen. Um eine durch <b>U</b> verursachte Änderung rückgängig zu machen, muß als nächstes das Kommando <b>u</b> eingegeben werden. |

**Beispiel**

```
$ print Hallo egon_
ESC $ print Hallo egon_
2b $ print _Hallo egon
dw $ print _egon
u $ print _Hallo egon
w $ print Hallo _egon
iLieber ESC $ print Hallo Lieber_egon
l $ print Hallo Lieber _egon
cwEmilESC $ print Hallo Lieber Emil_
U $ _print Hallo egon
dd $ _
```

## Sonstige Kommandos (Kommandomodus)

| Kommando                                                                          | Wirkung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | übergibt die momentane Zeile an die <b>ksh</b> zur Verarbeitung;  kann sowohl im Eingabemodus als auch im Kommandomodus eingegeben werden.                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>Strg-l</b>                                                                     | <i>(line redraw)</i> Cursor in die nächste Zeile bewegen und dort erneut die momentane Zeile anzeigen; wird meist verwendet, um eine Kommandozeile, die durch Steuerzeichen nicht mehr den wirklichen Text wiedergibt, bereinigt anzeigen zu lassen.                                                                                                                                                                                                                                                                                                                                                                        |
| <b>#</b>                                                                          | trägt die momentane Kommandozeile als Kommentar in die History-Datei ein.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>=</b>                                                                          | (Gleichheitszeichen) listet die Pfadnamen auf, die aus der Expandierung (bei Anhängen von <b>*</b> ) des Cursorwortes resultieren würden.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>\</b>                                                                          | hängt die fehlenden Zeichen an das <i>vi-WORT</i> unter dem Cursor an, um es zu einem Pfadnamen einer existierenden Datei zu vervollständigen. Angehängt werden dabei Zeichen bis zu einem Punkt, von dem ab mehrere Pfadnamen abgedeckt sind, oder bis der Pfadname vollständig ist. Wenn aus dieser Expandierung ein vollständiger Pfadname resultiert, so wird noch ein / angehängt, wenn es sich bei diesem Pfadnamen um ein Directory handelt, ansonsten wird ein Leerzeichen angehängt.<br><br>Diese Direktive <b>\</b> ist jedoch nur auf <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden. |
| <b>*</b>                                                                          | bewirkt, daß ein <b>*</b> an das <i>vi-WORT</i> unter dem Cursor angehängt wird, bevor dann Dateinamen-Expandierung für dieses Wort versucht wird. Wenn Expandierung für das Wort möglich ist, so wird das Wort durch den entsprechenden Dateinamen ersetzt und in den Eingabemodus umgeschaltet; ansonsten bleibt das Wort unverändert, und es wird ein Klingelton ausgegeben.                                                                                                                                                                                                                                             |
| <b>@buchstabe</b>                                                                 | durchsucht die Alias-Liste nach einem Alias-Namen <i>_buchstabe</i> . Falls ein solcher Alias-Name existiert, wird dessen Wert von der <b>ksh</b> an der Cursorposition eingefügt bzw. werden die darin enthaltenen <b>vi</b> -Direktiven ausgeführt.                                                                                                                                                                                                                                                                                                                                                                       |
| <b>~</b>                                                                          | Zeichen an der Cursorposition von Klein- in Großbuchstaben umwandeln bzw. umgekehrt; Cursor wird danach um eine Position nach rechts bewegt.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>n~</b>                                                                         | ab der Cursorposition <i>n</i> Zeichen von Klein- in Großbuchstaben umwandeln bzw. umgekehrt; Cursor wird danach um <i>n</i> Zeichen nach rechts bewegt.                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>.</b>                                                                          | Letztes Änderungskommando (eventuell auch aus einer vorherigen Kommandozeile) wiederholen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |



| Kommando  | Wirkung                                                                                                                                                                                                                                                                                                                                                        |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>n.</i> | Letztes Änderungskommando <i>n</i> mal wiederholen.                                                                                                                                                                                                                                                                                                            |
| <b>v</b>  | ( <b>vi</b> ) ruft den wirklichen Editor <b>vi</b> mit<br><br><code>fc -e \${VISUAL:-\${EDITOR:-vi}}</code><br><br>auf. In der zu editierenden Datei befindet sich dabei die momentane Kommandozeile, die nun mit dem wirklichen <b>vi</b> editiert werden kann. Nach dem Verlassen von <b>vi</b> wird diese Kommandozeile dann von der <b>ksh</b> ausgeführt. |
| <b>nv</b> | ( <b>vi</b> ) ruft den wirklichen Editor <b>vi</b> mit<br><br><code>fc -e \${VISUAL:-\${EDITOR:-vi}} n</code><br><br>auf. In der zu editierenden Datei befindet sich dabei die <i>n</i> .te Kommandozeile aus der History-Datei. Nach dem Verlassen von <b>vi</b> wird die editierte Kommandozeile dann von der <b>ksh</b> ausgeführt.                         |

**Beispiel**

```
$ print Hallo wie geht es emil_
ESC $ print Hallo wie geht es emil_
0 $ print Hallo wie geht es emil_
w $ print Hallo wie geht es emil_
12~ $ print hALLO WIE GEht es emil_
6h $ print hALLO WIE GEht es emil_
3~ $ print hALLO wie_GEht es emil_
. $ print hALLO wie geht es emil_
3b $ print hALLO wie geht es emil_
~ $ print HALLO wie geht es emil_
4. $ print Hallo_wie geht es emil_
0 $ print Hallo wie geht es emil_
dd $ _

$ alias _Q='0wi"Strg-vESCA"Strg-vESC' 
$ print Guten Morgen,\n lieber Emil_
Esc $ print Guten Morgen,\n lieber Emil_
@Q $ print "Guten Morgen,\n lieber Emil_"
Strg-m Ausgabe: Guten Morgen,
 lieber Emil
$ _
```

### Kommandozeilen aus der History-Datei holen (Kommandomodus)

Die nachfolgenden **vi**-Direktiven holen Kommandozeilen aus der History-Datei. Diese Direktiven wirken also nicht nur auf die aktuelle, sondern auch auf bereits früher eingegebene Kommandozeilen. Die Shell-Variable **HISTSIZE** legt dabei fest, wie viele der früheren Kommandozeilen in der History-Datei aufgehoben werden.

Ein **ksh**-Kommando kann sich über mehrere Zeilen erstrecken. Bei einem mehrzeiligen Kommando werden die Neuezeile-Zeichen (außer dem letzten) als ^J angezeigt.

| Kommando                                                                                            | Wirkung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>k</b><br>-                                                                                       | vorherige Kommandozeile aus der History-Datei holen. Eine erneute Eingabe von <b>k</b> oder - holt dann die Kommandozeile vor der vorherigen Kommandozeile usw.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>nk</b><br><i>n</i> -                                                                             | <i>n</i> .te vorherige Kommandozeile aus der History-Datei holen. Wenn <i>n</i> größer ist als die Anzahl der noch vorhandenen Kommandozeilen (zum Dateianfang hin), so wird ein Klingelton ausgegeben und die erste Kommandozeile aus der History-Datei geholt.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>j</b><br>+                                                                                       | nächste (nachfolgende) Kommandozeile aus der History-Datei holen. Eine erneute Eingabe von <b>j</b> oder + holt dann wieder die nachfolgende Kommandozeile usw.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>nj</b><br><i>n</i> +                                                                             | <i>n</i> .te nachfolgende Kommandozeile aus der History-Datei holen. Wenn <i>n</i> größer ist als die Anzahl der noch vorhandenen Kommandozeilen (zum Dateieende hin), so wird ein Klingelton ausgegeben und die letzte Kommandozeile aus der History-Datei geholt.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>G</b>                                                                                            | (Go back) älteste (erste) Kommandozeile aus der History-Datei holen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>nG</b>                                                                                           | (Go back) <i>n</i> .te Kommandozeile aus der History-Datei holen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>/string</i>   | sucht von links nach rechts in einer Zeile und zum Dateianfang hin in der History-Datei nach einem Vorkommen von <i>string</i> . Wird eine solche Zeile gefunden, so wird diese geholt.<br><br>Wird kein <i>string</i> angegeben ( <i>/</i>  ), so wird nach dem <i>string</i> aus dem letzten Suchkommando gesucht.<br><br>Unterschiede zwischen dem builtin- <b>vi</b> und dem wirklichen <b>vi</b> sind:<br><ul style="list-style-type: none"> <li>- keine regulären Ausdrücke im builtin-<b>vi</b> möglich</li> <li>- <i>/</i> und <i>?</i> wirken im builtin-<b>vi</b> genau in umgekehrter Richtung als im wirklichen <b>vi</b></li> <li>- im builtin-<b>vi</b> kann die Form <i>/string/+n</i> nicht verwendet werden. Anders als im wirklichen <b>vi</b> wird beim Suchen im builtin-<b>vi</b> nicht automatisch vom Dateianfang auf Dateieende bzw. vom Dateieende auf Dateianfang umgeschaltet.</li> <li>- Im builtin-<b>vi</b> kann der <i>string</i> auch die Zeichen <i>/</i> oder <i>?</i> enthalten.</li> </ul> |
| <i>/^string</i>  | dasselbe wie <i>/string</i>  , außer daß <i>string</i> nur gefunden wird, wenn er am Anfang einer Zeile steht; nur auf <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

| Kommando                                                                                                | Wirkung                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>?string</code>   | dasselbe wie <code>/string</code>  , außer daß in umgekehrter Richtung (in einer Zeile von rechts nach links und vorwärts zum History-Dateiende hin) gesucht wird.                                            |
| <code>?^string</code>  | dasselbe wie <code>?string</code>  , außer daß <code>string</code> nur gefunden wird, wenn er am Anfang einer Zeile steht; nur auf <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden. |
| <b>n</b>                                                                                                | letztes Suchkommando ( <code>/</code> oder <code>?</code> ) wiederholen.                                                                                                                                                                                                                       |
| <b>N</b>                                                                                                | letztes Suchkommando ( <code>/</code> oder <code>?</code> ) mit umgekehrter Suchrichtung wiederholen.                                                                                                                                                                                          |

**Beispiel**

```
$ set -o vi 
$ PS1="! $ " 
160 $ date 
Tue Jun 11 09:13:23 MESZ 1996
161 $ print $OLDPWD 
/user1/egon
162 $ alias type="cat " 
163 $ for i in b* 
> do wc -l $i 
> done 
 1 basisname
 10 bname
164 $ find $HOME -name isnum -print 
/user1/egon/flib/isnum
165 $ history -5 
160 date
161 print $OLDPWD
162 alias type="cat "
163 for i in b*
 do wc -l $i
 done
164 find $HOME -name isnum -print
165 history -5
166 $
ESC 166 $ _ [um vom Eingabe- in Kommandomodus
 umzuschalten]

k 166 $ history -5
- 166 $ find $HOME -name isnum -print
- 166 $ for i in b^Jdo wc -l $i^Jdone
Strg-m 1 basisname
 10 bname
ESC 167 $ _ [um vom Eingabe- in Kommandomodus
 umzuschalten]

k 167 $ for i in b^Jdo wc -l $i^Jdone
/cat  167 $ alias type="cat "
/da  167 $ date
```

```
+ 167 $ print $OLDPWD
?^fin[↵] 167 $ find $HOME -name isnum -print
3W 167 $ find $HOME -name isnum -print
dw 167 $ find $HOME -name _print
i"ein*" ESC 167 $ find $HOME -name "ein*" _print
[↵] /user1/egon/kshueb/eingabe
 /user1/egon/kshueb/einles
 168 $
```

## 5.21 Fehlersuche in ksh-Skripts

Die Debugging-Möglichkeiten der **ksh** wurden gegenüber denen der Bourne-Shell verbessert.

Auch die **ksh** bietet fünf Optionen zur Fehlersuche in Skripts an:

| Option    | Optionsname <sup>a</sup> | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-n</b> | <b>noexec</b>            | Kommandos werden nur gelesen und auf Syntaxfehler untersucht, aber nicht ausgeführt.                                                                                                                                                                                                                                                                                                                   |
| <b>-v</b> | <b>verbose</b>           | Alle Shell-Eingabezeilen werden so ausgegeben, wie sie gelesen werden. Erst nach dieser Ausgabe wird das entsprechende Kommando ausgeführt.                                                                                                                                                                                                                                                            |
| <b>-x</b> | <b>xtrace</b>            | Für jede Shell-Eingabezeile wird zuerst Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung durchgeführt. Danach wird die so entstandene Kommandozeile auf die Standardfehlerausgabe ausgegeben; vor dieser Kommandozeile wird dabei noch der Inhalt der Shellvariablen <b>PS4</b> <sup>b</sup> angegeben. Erst nach dieser Ausgabe wird die betreffende Kommandozeile ausgeführt. |
| <b>-u</b> | <b>nounset</b>           | Ein Zugriff auf nicht gesetzte Variablen bewirkt einen Fehler. Ohne diese Option wird üblicherweise die leere Zeichenkette geliefert.                                                                                                                                                                                                                                                                  |
| <b>-e</b> | <b>errexit</b>           | Wenn ein Kommando einen exit-Status verschieden von 0 (nicht erfolgreich) liefert, dann führt die <b>ksh</b> die für das Signal <b>ERR</b> <sup>c</sup> installierte Signalbehandlung aus und beendet sich dann.                                                                                                                                                                                       |

- a. Die Verwendung von Optionsnamen wird in Kapitel 5.24.3 gezeigt.
- b. Die Variable **PS4** (siehe Kapitel 5.8.4) ist nur auf nach dem 3.6.1986 freigegebenen **ksh**-Versionen verfügbar. Frühere **ksh**-Versionen geben anstelle von **PS4** das Zeichen + (wie die Bourne-Shell) aus.
- c. Siehe Kapitel 5.22.

Diese Optionen können zunächst wieder auf zwei Arten gesetzt werden:

**ksh [-nvxue] skript**

In diesem Fall gelten die gesetzten Optionen nur für die Dauer der Skript-Ausführung.

**set [-nvxue]**

Hierbei bleiben die entsprechenden Optionen solange gesetzt, bis sie explizit mit **+** wieder ausgeschaltet werden:

**set [+nvxue]**

Im Unterschied zur vorherigen Aufrufform werden dabei die Optionen nicht für ein aufgerufenes Shell-Skript, das ja von einer eigenen Subshell ausgeführt wird, gesetzt, sondern gelten für die gerade aktive Shell.

Neu in der **ksh** ist, daß die entsprechenden Optionen auch über die oben fett gedruckten Optionsnamen gesetzt werden können:

**-o optname** Einschalten der Option *optname*

**+o optname** Ausschalten der Option *optname*<sup>1</sup>

Somit können die entsprechenden Optionen auch mit den folgenden beiden Aufrufformen ein- bzw. ausgeschaltet werden:

**ksh [**±o** optname] skript****set [**±o** optname]**

Da die Option **-x** bzw. **xtrace** die am häufigsten verwendete Debug-Option ist, wird diese hier genauer behandelt.

Um sich beim **xtrace** immer die Zeilennummer mit ausgeben zu lassen, müßte im Debug-Prompt **PS4** auf die Shellvariable **LINENO** zugegriffen werden, z.B.

```
PS4='Zeile $LINENO: ' 2
```

Soll auch die Ausführung von Funktionen zeilenweise angezeigt werden, so kann dies mit

**typeset -ft**

erreicht werden.

---

1. Siehe auch Kapitel 5.24.3.

2. Die beiden Variablen **PS4** und **LINENO** sind nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden. Es ist empfehlenswert, die folgenden Zeilen oder zumindest ähnliche in der Datei **\$HOME/.profile** einzutragen:

```
PS4='Zeile $LINENO: '
export PS4
```

In der Environment-Datei könnte z.B. eine Funktion mit Namen *breakpoint* definiert werden:

```
typeset -fx breakpoint

function breakpoint
{
 typeset zeile
 while true
 do echo -n "Zeile $lineno :ksh-dbg>> "
 read -r zeile
 if [[-z "$zeile"]]
 then return
 fi
 eval "$zeile"
 done
}
```

An jeder kritischen Stelle in einem Shell-Skript könnte nun diese Funktion *breakpoint* aufgerufen werden, um während des Debuggens eines **ksh**-Skripts interaktiv Eingaben zu ermöglichen. Über diese interaktiven Eingaben wäre es dann z.B. möglich, sich die momentanen Werte von Variablen ausgeben zu lassen.

Auf **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, wird noch das Signal **DEBUG**<sup>1</sup> angeboten. Dieses Signal wird automatisch nach jeder Kommando-Ausführung geschickt. Mit **trap** könnte man dann erreichen, daß die Funktion *breakpoint* nach jedem Kommando automatisch aufgerufen wird:

```
trap 'lineno=$LINENO;breakpoint' DEBUG2
```

Mit

```
trap 'lineno=$LINENO;breakpoint' ERR
```

würde *breakpoint* bei jedem Auftreten des Signals **ERR** (Ausführung eines Kommandos war fehlerhaft) automatisch aufgerufen.

## 5.22 Signalbehandlung in der ksh

In der **ksh** gilt für Signale im allgemeinen das gleiche wie in der Bourne-Shell<sup>3</sup>.

Neu in der **ksh** ist, daß die Signale nicht nur über ihre Nummern, sondern auch über symbolische Namen angesprochen werden können. Die symbolischen Namen der am häufigsten verwendeten Signale sind:

- 
1. Siehe Kapitel 5.22.
  2. **lineno=\$LINENO** ist notwendig, da **breakpoint** auf die globale Variable **lineno** und nicht auf die automatische Variable **LINENO** zugreift. Der Grund dafür ist, daß bei jedem Funktionsaufruf, also auch beim Aufruf von **breakpoint**, die automatische Variable **LINENO** automatisch auf 1 gesetzt wird.
  3. Siehe Kapitel 4.14.

| Name         | Bedeutung                                                                                                                       |
|--------------|---------------------------------------------------------------------------------------------------------------------------------|
| <b>HUP</b>   | <i>hangup</i> : wird beim Beenden einer Verbindung (z.B. Auflegen des Telefonhörers) erzeugt.                                   |
| <b>INT</b>   | <i>intr</i> : Interrupt-Signal, welches durch Drücken der <i>DEL</i> - oder <i>BREAK</i> -Taste ( <i>Strg-c</i> ) erzeugt wird. |
| <b>QUIT</b>  | <i>quit</i> : wird durch die Eingabe von <i>Strg-\</i> erzeugt.                                                                 |
| <b>TERM</b>  | <i>terminate</i> : wird von einem Prozeß an alle Prozesse eines Benutzers geschickt, um diese zu beenden.                       |
| <b>KILL</b>  | <i>kill</i> : bewirkt die sofortige Beendigung eines Prozesses und kann nicht abgefangen werden.                                |
| <b>EXIT</b>  | <i>exit</i> : wird beim Verlassen einer Funktion oder einer <b>ksh</b> erzeugt.                                                 |
| <b>ERR</b>   | <i>error</i> : wird immer dann geschickt, wenn ein Kommando einen exit-Status verschieden von 0 (nicht erfolgreich) liefert.    |
| <b>DEBUG</b> | <i>debugging</i> : wird nach jeder Ausführung eines Kommandos erzeugt.                                                          |

Für die Job-Kontrolle in der **ksh**<sup>1</sup> stehen die folgenden Signale zur Verfügung:

| Name        | Bedeutung                                                                                                                                                                                                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>TSTP</b> | <b>tty stop</b> : wird allen Vordergrund-Prozessen eines Benutzers geschickt, wenn die Taste <i>Suspend</i> (normalerweise <i>Strg-z</i> ) gedrückt wird. Die Ausführung der entsprechenden Vordergrund-Prozesse wird durch dieses Signal angehalten, wenn sie es nicht explizit abfangen. |
| <b>TTIN</b> | <b>tty input</b> : wird jedem Hintergrund-Prozeß geschickt, der vom Terminal zu lesen versucht. Die Ausführung des entsprechenden Hintergrund-Prozesses wird durch dieses Signal angehalten, wenn dieser es nicht explizit abfängt.                                                        |
| <b>CONT</b> | <b>continue</b> : wird dieses Signal einem angehaltenen Prozeß geschickt, so wird dessen Ausführung fortgesetzt.                                                                                                                                                                           |
| <b>STOP</b> | <b>stop</b> : Dieses Signal bewirkt, daß die Ausführung des Prozesses, dem es geschickt wird, angehalten wird. Mit dem Signal <b>CONT</b> kann dann an späterer Stelle dessen Ausführung wieder fortgesetzt werden. Das Signal <b>STOP</b> kann niemals von einem Prozeß ignoriert werden. |

Tasten, die das Schicken von Signalen an Vordergrund-Prozesse bewirken, können mit **stty** auch anders gesetzt werden.

---

1. Siehe Kapitel 5.23.

Die Signalnummern können von System zu System verschieden sein. Um sich zu allen verfügbaren Signalen deren Nummern und Namen ausgeben zu lassen, steht das Kommando

### **kill -l**

zur Verfügung.

#### **Beispiel**

```
$ kill -l 
```

|          |              |
|----------|--------------|
| 1) HUP   | 15) TERM     |
| 2) INT   | 16) USR1     |
| 3) QUIT  | 17) USR2     |
| 4) ILL   | 18) CHLD     |
| 5) TRAP  | 19) PWR      |
| 6) IOT   | 20) WINCH    |
| 7) EMT   | 21) bad trap |
| 8) FPE   | 22) POLL     |
| 9) KILL  | 23) STOP     |
| 10) BUS  | 24) TSTP     |
| 11) SEGV | 25) CONT     |
| 12) SYS  | 26) TTIN     |
| 13) PIPE | 27) TTOU     |
| 14) ALRM |              |

```
$
```

Zum Abfangen von Signalen steht wie in der Bourne-Shell das builtin-Kommando **trap** zur Verfügung:

**trap** [*argument*] [*signal(e)*]

Das Kommando **trap** legt die Reaktion der **ksh** für asynchron eintreffende Signale fest. Trifft eines der angegebenen *signal(e)* ein, dann führt die **ksh** die als *argument* angegebene Kommandoliste aus und danach setzt sie ihre Ausführung an der Stelle fort<sup>1</sup>, an der die durch das Signal bedingte Unterbrechung stattfand.

Als Signalbehandlung ist dabei wieder (wie in der Bourne-Shell) möglich:

1. Es kann für *argument* eine Liste von Kommandos angegeben werden ('*kdoliste*'), die bei Eintreffen eines der *signal(e)* auszuführen ist.
2. Ignorieren der *signal(e)*, indem für *argument* eine leere Zeichenkette (z.B. "" oder "") angegeben wird.
3. Wird beim Aufruf von **trap** kein *argument* oder für *argument* nur – (Minuszeichen) angegeben, so wird für die *signal(e)* wieder die vom System voreingestellte Signalbehandlung festgelegt.

Werden beim Aufruf von **trap** kein *argument* und keine *signal(e)* angegeben, dann gibt **trap** die Signalnummern bzw. Signalnamen aus, für die momentan mit Hilfe eines **trap**-Kommandos eine benutzerspezifische Signalbehandlung eingestellt

---

1. Wenn bei den für *argument* angegebenen Kommandos nicht **exit** vorkommt.



ist; zudem gibt es in diesem Fall zu jeder dieser Signalnummern die Kommandos an, die diese Signalbehandlung durchführen.

Für *signal(e)* kann folgendes angegeben werden:

- ▶ **Name oder Nummer eines Signals**  
Die speziellen Signalnummern und Signalnamen können von System zu System verschieden sein. Mit **kill -l** können alle Signalnummern und Signalnamen am jeweiligen System aufgelistet werden. Aus Portabilitätsgründen ist es ratsam, Signalnamen (wenn möglich) anstelle von Signalnummern zu verwenden.
- ▶ **ERR**  
Die **ksh** führt die als *argument* angegebenen Kommandos immer dann aus, wenn ein Kommando einen exit-Status verschieden von 0 (nicht erfolgreich) liefert. Eine mit **trap** für dieses Signal installierte Signalbehandlung wird nicht an eine Funktion vererbt.
- ▶ **0 oder EXIT**  
Wird das **trap**-Kommando für diese Signale innerhalb einer Funktion ausgeführt, dann werden die als *argument* angegebenen Kommandos beim Verlassen der Funktion ausgeführt.  
Wurde dagegen das **trap**-Kommando außerhalb einer Funktion ausgeführt, dann werden die als *argument* angegebenen Kommandos beim Verlassen der **ksh** ausgeführt.
- ▶ **DEBUG**  
Die **ksh** führt die als *argument* angegebenen Kommandos nach jedem einfachen Kommando aus. Eine mit **trap** für dieses Signal eingerichtete Signalbehandlung wird nicht an eine Funktion vererbt.  
Das **DEBUG**-Signal ist nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.

**Beispiel****trap**

Auflisten aller benutzerspezifischen Signalhandler

**trap '\$HOME/.logout' EXIT**

Die Kommandos aus der Datei *.logout* werden unmittelbar vor dem Verlassen der **ksh** ausgeführt.

**trap - INT QUIT EXIT**

Die Signalbehandlung für die Signale **INT**, **QUIT** und **EXIT** wird auf die vom System voreingestellte Signalbehandlung zurückgesetzt.

**trap INT QUIT EXIT**

gleiche Auswirkung wie der vorherige Aufruf.

**Hinweise**

- ▶ Die bei **trap** als *argument* angegebene Kommandoliste wird zweimal gelesen: das erstmal bei der Ausführung des **trap**-Kommandos und das zweitemal, wenn die Kommandoliste bedingt durch das Eintreffen eines Signals aufgerufen wird. Deshalb empfiehlt es sich, die für *argument* angegebene Kommandoliste mit `'..'` zu klammern, um Parametersubstitution, Kommandosubstitution oder Dateinamen-Expandierung beim erstmaligen Lesen auszuschalten.
- ▶ Wenn mehrere Signale zur gleichen Zeit auftreten, dann arbeitet die **ksh** diese in der folgenden Reihenfolge ab:
  1. **DEBUG**
  2. **ERR**
  3. andere Signale in der Reihenfolge, wie sie durch die Signalnummern vorgegeben ist.
  4. **EXIT** (immer als letztes)

Eine **ksh**-Subshell ignoriert alle Signale, welche von ihrem Elternprozeß (Elternshell) ignoriert werden.

**Beispiel**

```
$ trap "echo DEBUG-trap" DEBUG ↵
DEBUG-trap
$ trap "echo ERR-trap" ERR ↵
DEBUG-trap
$ eco ↵
ksh: eco: not found
DEBUG-trap
ERR-trap
$ trap '' DEBUG ERR ↵ [DEBUG und ERR ignorieren]
$
```

## 5.23 Job-Kontrolle in der Korn-Shell

Jede Pipeline<sup>1</sup>, die man ausführen läßt, wird als Job bezeichnet. Die **ksh** verfügt anders als die Bourne-Shell über eine eigene Job-Kontrolle, mit der es möglich ist, Jobs anzuhalten und wieder fortzusetzen, oder die Ausführung von Jobs vom Hintergrund in den Vordergrund zu verlagern bzw. umgekehrt.

---

1. Als Erinnerung: Eine Pipeline kann auch nur ein Kommando sein; siehe Kapitel 4.4.2.

### 5.23.1 Allgemeines

Um die Job-Kontrolle der **ksh** zu aktivieren, muß die Option **monitor**<sup>1</sup> gesetzt werden. Dies ist allerdings systemabhängig:

- ▶ Auf Systemen, welche die vollständige Job-Kontrolle anbieten, wird die Option **monitor** automatisch gesetzt, wenn eine **ksh** als interaktive Shell aufgerufen wird.
- ▶ Auf anderen Systemen muß explizit **set -o monitor** aufgerufen werden; diesen Aufruf schreibt man dort üblicherweise in die Environment-Datei.

Wenn die Option **monitor** gesetzt ist, so gibt die **ksh** bei jedem Start eines Hintergrund-Jobs dessen *[Jobnummer]* und *PID* aus.

Wird ein Hintergrund-Job beendet, so meldet die **ksh** dies mit *[Jobnummer]* + *Done* *Jobname* (*Kommandozeile des Jobs*).

```
Beispiel $ cat /usr/include/* -print | wc -c &
[1] 356
$ ps
 PID TTY TIME COMMAND
 255 01 0:06 ksh
 356 01 0:01 wc
 357 01 0:01 cat
 358 01 0:00 ps
322908 [Ausgabe des Hintergrund-Jobs]

[1] + Done cat /usr/include/* | wc -c &
$
```

Die einzelnen Jobs können über die PID, die Jobnummer oder den Jobnamen angesprochen werden. Dazu stehen die builtin-Kommandos **wait**, **kill**, **fg** und **bg** zur Verfügung. Um einen Job über seine Jobnummer oder seinen Namen anzusprechen, gibt es folgende Notationen:

|                   |                                            |
|-------------------|--------------------------------------------|
| <b>%jobnr</b>     | Job mit Jobnummer <i>jobnr</i>             |
| <b>%string</b>    | Job, dessen Name mit <i>string</i> beginnt |
| <b>%?string</b>   | Job, dessen Name den <i>string</i> enthält |
| <b>%+ oder %%</b> | momentaner Job                             |
| <b>%-</b>         | vorheriger Job                             |

---

1. Siehe Kapitel 5.24.3.

Mit dem builtin-Kommando **jobs** können die gerade im Hintergrund ablaufenden Jobs und deren momentaner Status angezeigt werden.

Systeme, die eine vollständige Job-Kontrolle anbieten, ermöglichen es, einen Prozeß anzuhalten und ihn dann vom Vordergrund in den Hintergrund zu verlagern bzw. umgekehrt. Der Job, der gerade im Vordergrund abgearbeitet wird, kann mit der *Susp*-Taste<sup>1</sup> (meist *Strg-z*) angehalten werden. Die **ksh** meldet dann, daß der entsprechende Job angehalten wurde und zeigt mit der Ausgabe des Prompts an, daß sie nun wieder für die Entgegennahme von Kommandos bereit ist. Allerdings können niemals Funktionen oder Programmiersprach-Kommandos<sup>2</sup> angehalten werden.

Ein Hintergrund-Job wird immer dann automatisch angehalten, wenn er versucht, vom Terminal zu lesen. Mit dem Kommando **stty tostop**<sup>3</sup> kann zusätzlich noch festgelegt werden, daß Hintergrund-Jobs auch dann angehalten werden, wenn sie versuchen, auf das Terminal zu schreiben. Die Ausführung solcher Jobs wird dann erst wieder fortgesetzt, wenn diese in den Vordergrund verlagert werden.

Nachfolgend werden nun noch die für die Job-Kontrolle benötigten builtin-Kommandos genauer vorgestellt.

## 5.23.2 Informationen zu Hintergrund-Jobs (jobs)

**jobs** [-lp] [*job(s)*]

Mit dem builtin-Kommando **jobs** können Informationen zu den angegebenen *job(s)* oder allen momentan aktiven Jobs (wenn *job(s)* nicht angegeben ist) abgefragt werden.

Die **ksh** gibt dabei zu jedem einzelnen Job eine Zeile aus, in der folgende Information enthalten ist:

[*Jobnummer*]

mit einem + vor dem aktuellen Job bzw. einem - (Minuszeichen) vor dem vorherigen aktuellen Job<sup>4</sup>. Aktueller Job ist der Job, der zuletzt im Hintergrund gestartet wurde und der vorherige aktuelle Job wurde als vorletzter Hintergrund-Job gestartet.

---

1. Mit dem Aufruf

**stty**

kann festgestellt werden, auf welche Taste *susp* eingestellt ist.

Mit

**stty susp ^z**

könnte *susp* auf *Strg-z* eingestellt werden.

2. Siehe Kapitel 5.16.

3. Dies ist ab System V.2 nicht mehr notwendig.

4. + und - können eventuell auch nach der Jobnummer angegeben sein.

## Status

| Bezeichnung       | Bedeutung                       |
|-------------------|---------------------------------|
| <i>Running</i>    | befindet sich in der Ausführung |
| <i>Stopped</i>    | ist momentan angehalten         |
| <i>Done</i>       | wurde normal beendet            |
| <i>Terminated</i> | wurde abgebrochen               |

(*Nummer*) nach *Done*

*Nummer* gibt dabei den exit-Status dieses beendeten Jobs an; wird jedoch nur angezeigt, wenn der exit-Status verschieden von 0 ist.

Kommandozeile (Jobname)

gibt die Kommandozeile wieder, die zum Start des Jobs führte. Die **ksh** nimmt dabei die entsprechende Kommandozeile aus der History-Datei. Falls die **ksh** nicht auf die History-Datei zugreifen kann, dann wird keine Kommandozeile angezeigt.

## Optionen:

| Option    | Bedeutung                                                            |
|-----------|----------------------------------------------------------------------|
| <b>-l</b> | nach der Jobnummer wird zusätzlich noch die PID des Jobs ausgegeben. |
| <b>-p</b> | Es werden nur die PIDs der Jobs angezeigt.                           |

Der exit-Status von **jobs** ist immer 0 (erfolgreich).

### Beispiel

```
$ jobs -l
+[4] 239 Running cc -c addiere.c &
-[3] 452 Stopped mail eml
[2] 632 Done (1) find / -name "*.c" -print | wc -l &
$ jobs -p %cc
239
$
```

### 5.23.3 Signale an Jobs schicken (kill)

**kill** [-signal] job(s) (1)

oder

**kill -l** (2)

Mit dem builtin-Kommando **kill** wird den angegebenen job(s) das Signal *signal* geschickt. Dieses geschickte Signal kann von den entsprechenden job(s) entweder ignoriert oder mit einem installierten Signalhandler abgefangen werden, andernfalls werden die betreffenden job(s) durch diesen Aufruf abgebrochen.

Für *signal* kann entweder eine Signalnummer oder ein Signalname<sup>1</sup> angegeben werden.

Ist -*signal* nicht angegeben, dann wird das Signal **TERM** geschickt.

Falls einem angehaltenen Job das Signal **TERM** oder **HUP** geschickt wird, so sendet die **ksh** diesem Job zuvor das Signal **CONT**.

Um Hintergrund-Jobs anzuhalten, muß diesen das **STOP**-Signal geschickt werden:

**kill -STOP ...**

Es ist im übrigen empfehlenswert, ein Alias der Form

**alias stop='kill -STOP'**<sup>2</sup>

zu definieren, um die Tipparbeiten beim Anhalten von Hintergrund-Jobs etwas zu reduzieren.

Der exit-Status des Aufrufs (1) ist die Anzahl von Prozessen, welchen **kill** nicht erfolgreich das entsprechende Signal schicken konnte.

Um sich alle auf dem jeweiligen System verfügbaren Signalnummern und Signalnamen anzeigen zu lassen, steht die zweite Aufrufform (2) zur Verfügung:

**kill -l**

Der exit-Status dieses Aufrufs (2) ist immer 0 (erfolgreich).

---

1. Siehe Kapitel 5.22.

2. Auf manchen **ksh**-Versionen ist dieses Alias auch bereits vordefiniert.

### 5.23.4 Auf die Beendigung von Jobs warten (**wait**)

**wait** [*job(s)*]

Das Kommando **wait** veranlaßt die **ksh** auf die Beendigung der von den angegebenen *job(s)* gestarteten Prozesse zu warten.

Für *job(s)* können allerdings auch die PIDs von Prozessen angegeben werden, um auf deren Beendigung zu warten<sup>1</sup>.

Falls keine *job(s)* angegeben sind, so wird auf die Beendigung aller Kindprozesse der momentan aktiven **ksh** gewartet.

Der exit-Status von **wait** ist der exit-Status des letzten Prozesses, auf den gewartet wurde.

|                 |                  |                                                              |
|-----------------|------------------|--------------------------------------------------------------|
| <b>Beispiel</b> | <b>wait</b>      | Auf die Beendigung aller Hintergrund-Prozesse warten.        |
|                 | <b>wait %3</b>   | Auf die Beendigung des Jobs mit der Jobnummer 3 warten.      |
|                 | <b>wait 1462</b> | Auf die Beendigung des Prozesses mit der PID 1462 warten.    |
|                 | <b>wait \$!</b>  | Auf die Beendigung des letzten Hintergrund-Prozesses warten. |

### 5.23.5 Ausführung von angehaltenen Jobs im Hintergrund fortsetzen (**bg**)

**bg** [*job(s)*]

**bg** bewirkt, daß die Ausführung der angehaltenen *job(s)* im Hintergrund fortgesetzt wird. Werden keine *job(s)* beim Aufruf angegeben, so wird die Ausführung des aktuellen Jobs (zuletzt angehaltener Job) im Hintergrund fortgesetzt. Der aktuelle Job kann auch mit dem Kommando **jobs** ermittelt werden.

Der exit-Status von **bg** ist 0, wenn die Option **monitor** gesetzt ist, ansonsten ist er 1.

|                |                                                                                          |
|----------------|------------------------------------------------------------------------------------------|
| <b>Hinweis</b> | <b>bg</b> ist nur auf Systemen ein builtin-Kommando, welche über Job-Kontrolle verfügen. |
|----------------|------------------------------------------------------------------------------------------|

---

1. In der **ksh**-Version vom 3.6.1986 und früheren Versionen können für *job(s)* sogar nur PIDs angegeben werden.

### 5.23.6 Ausführung von Hintergrund-Jobs im Vordergrund fortsetzen (fg)

**fg** [job(s)]

**fg** bewirkt, daß die Ausführung der angegebenen Hintergrund-job(s) der Reihe nach im Vordergrund fortgesetzt wird. Werden keine job(s) beim Aufruf angegeben, so wird die Ausführung des aktuellen Jobs (zuletzt angehaltener Job oder zuletzt im Hintergrund gestarteter Job) im Vordergrund fortgesetzt. Der aktuelle Job kann auch mit dem Kommando **jobs** ermittelt werden.

Der exit-Status von **fg** ist 0, wenn die Option **monitor** gesetzt ist, ansonsten ist er 1.

**Beispiel**

```
$ stty susp ^z
$ cat teiler
if [[$# -eq 0]]
then echo "usage: $0 zahl(en)"
 exit 1
fi

integer i

for zahl
do echo -n "$zahl ist "
 if isnum $zahl
 then
 echo -n "teilbar durch: "
 i=1
 while ((i<=zahl))
 do if ((zahl%i==0))
 then echo -n " $i"
 fi
 i=i+1
 done
 echo
 else
 echo "keine erlaubte Zahl"
 fi
done
$ chmod u+x teiler
$ teiler 1234567 45 3.4 45673 >teiler.txt
[Strg-z]
+ [1] Stopped teiler 1234567 45 3.4 45673 >teiler.txt
$ teiler 222225 >teiler1.txt
[Strg-z]
+ [2] Stopped teiler 222225 >teiler1.txt
$ jobs
+ [2] Stopped teiler 222225 >teiler1.txt
- [1] Stopped teiler 1234567 45 3.4 45673 >teiler.txt
```



```

$ bg
[2] teiler 2222225 >teiler1.txt&
$ jobs
+ [2] Running teiler 2222225 >teiler1.txt
- [1] Stopped teiler 1234567 45 3.4 45673 >teiler.txt
$ stop %2
$ jobs
+ [2] Stopped(signal) teiler 2222225 >teiler1.txt
- [1] Stopped teiler 1234567 45 3.4 45673 >teiler.txt
$ bg %1
[1] teiler 1234567 45 3.4 45673 >teiler.txt&
$ jobs
+ [2] Stopped (signal) teiler 2222225 >teiler1.txt
- [1] Running teiler 1234567 45 3.4 45673 >teiler.txt
$ fg
teiler 2222225 >teiler1.txt
[Strg-z]
+ [2] Stopped teiler 2222225 >teiler1.txt
$ bg %-
- [1] teiler 1234567 45 3.4 45673 >teiler.txt&
$ fg
teiler 2222225 >teiler1.txt
[Strg-z]
+ [2] Stopped teiler 2222225 >teiler1.txt
$ bg
[2] teiler 2222225 >teiler1.txt&
$ jobs
+ [2] Running teiler 2222225 >teiler1.txt
- [1] Running teiler 1234567 45 3.4 45673 >teiler.txt
$ alias %=fg
$ function _fg
> {
> fg %${1-%}
> }
$ %=
teiler 2222225 >teiler1.txt
[Strg-z]
+ [2] Stopped teiler 2222225 >teiler1.txt
$ bg
[2] teiler 2222225 >teiler1.txt&
$ % 1
teiler 1234567 45 3.4 45673 >teiler.txt
[Strg-z]
+ [1] Stopped teiler 1234567 45 3.4 45673 >teiler.txt
$ kill %1 %2
$ jobs
+ [1] Terminated teiler 1234567 45 3.4 45673 >teiler.txt
+ [2] Terminated teiler 2222225 >teiler1.txt
$ jobs
$

```

**Hinweis** **fg** ist nur auf Systemen ein builtin-Kommando, die über Job-Kontrolle verfügen.

## 5.23.7 Allgemeine Hinweise zur Job-Kontrolle

Beim Versuch, eine interaktive **ksh** mit **exit** (bzw. *Strg-D*) zu verlassen, während noch Jobs im Hintergrund ablaufen, wird eine Meldung ausgegeben und die **ksh** nicht verlassen. Wird aber unmittelbar danach nochmals versucht, die **ksh** zu verlassen, dann wird sie trotz noch aktiver Jobs verlassen; die Ausführung der entsprechenden Hintergrund-Jobs wird dann jedoch beendet.

Wie bereits zuvor erwähnt, wird ein Hintergrund-Job, der versucht, von der Dialogstation zu lesen oder auf diese zu schreiben, von der **ksh** angehalten. Die **ksh** gibt dabei eine entsprechende Meldung aus:

```
[jobnr] + Stopped (tty input) kdo &
oder
```

```
[jobnr] + Stopped (tty output) kdo &
```

Der betreffende Job kann dann zu einem passenden Zeitpunkt mit **fg** in den Vordergrund gebracht werden, um ihm seine Aus- und Eingaben zu gestatten. Nachdem dies erledigt ist, könnte dieser Job mit

*Strg-Z* und danach mit

**bg**

wieder in den Hintergrund verlagert werden.

### Beispiel

```
$ stty tostop
$ cat eingabe
echo "Gib was ein:"
read
echo "Du hast ---$REPLY--- eingegeben"
$ eingabe &
[1] 292
[1] + Stopped(tty output) eingabe &
$ fg
eingabe
Gib was ein:
Hallo
Du hast ---Hallo--- eingegeben
$
```

## 5.24 Builtin-Kommandos der Korn-Shell

Die builtin-Kommandos sind Teile des Programms **ksh**. Deswegen muß die Korn-Shell keinen neuen Prozeß starten, um diese ablaufen zu lassen. Das hat zur Folge, daß diese Kommandos auch »schneller starten«<sup>1</sup> als die anderen (nicht builtin-) Kommandos.

Während die meisten dieser Kommandos wichtige Werkzeuge für Shell-Skripts sind, ist es für andere von Wichtigkeit, daß sie in der momentan aktiven Shell und auf keinen Fall in einer Subshell ablaufen.

Zunächst werden in diesem Kapitel alle noch nicht behandelten builtin-Kommandos der **ksh** ausführlich besprochen, bevor am Ende eine kurze Zusammenfassung aller builtin-Kommandos der **ksh** gegeben wird.

### 5.24.1 Das builtin-Kommando **cd**

Anders als in der Bourne-Shell kann das Kommando **cd** in der **ksh** auf zwei verschiedene Arten aufgerufen werden:

**cd** [*directory*]                      (1)

**cd** *altstring neustring*            (2)

Bei der ersten Aufrufform ist folgendes zu unterscheiden:

| <i>directory</i>                          | <i>Wirkung</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| nicht angegeben                           | Wechsel zum Home-Directory, das über die Shell-Variable <b>HOME</b> festgelegt ist.                                                                                                                                                                                                                                                                                                                                                                                                               |
| <i>/pfadname</i>                          | Wechsel zum Directory <i>/pfadname</i> ; Shell-Variable <b>CDPATH</b> wird hierbei nicht benutzt.                                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>pfadname</i> , der nicht mit / beginnt | Wenn <b>CDPATH</b> gesetzt ist und <i>pfadname</i> beginnt nicht mit ./ oder ../, dann sucht die <b>ksh</b> in den von <b>CDPATH</b> festgelegten Directories nach einem Subdirectory <i>pfadname</i> . Wird ein solches gefunden, dann wird dies das neue Working-Directory; in diesem Fall wird auch noch der Pfadname des neuen Working-Directorys ausgegeben.<br>Ist dagegen <b>CDPATH</b> nicht gesetzt, so wird <i>pfadname</i> (relativ zum Working-Directory) das neue Working-Directory. |

1. Siehe auch Kapitel 5.25, wo die Abarbeitung von Kommandozeilen gezeigt wird.

| <i>directory</i>                     | <i>Wirkung</i>                                                                                                                                                                                                        |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -                                    | (Minuszeichen) Die <b>ksh</b> macht das vorherige Working-Directory wieder zum Working-Directory und gibt dann den Pfadnamen dieses neuen Working-Directorys aus.<br><b>cd -</b> entspricht also <b>cd \$OLDPWD</b> . |
| existiert nicht oder fehlende Rechte | Die <b>ksh</b> gibt eine Fehlermeldung aus.                                                                                                                                                                           |

**cd** bewirkt unter anderem, daß die automatischen Variablen **PWD** (mit Pfadnamen des neuen Working-Directorys) und **OLDPWD** (mit Pfadnamen des vorherigen Working-Directorys) gesetzt werden.

Die zweite Aufrufform

**cd** *altstring neustring*

bewirkt, daß im Pfadnamen des momentanen Working-Directorys (**\$PWD**) der String *altstring* durch *neustring* ersetzt wird; danach versucht **cd** diesen durch Ersetzung entstandenen Pfadnamen zum neuen Working-Directory zu machen. Ist dies möglich, dann wird der Pfadname des neuen Working-Directorys ausgegeben.

Der exit-Status von **cd** ist bei beiden Aufrufformen 0, wenn ein Directory-Wechsel erfolgreich durchgeführt werden konnte, ansonsten verschieden von 0.

**Beispiel**

```
$ pwd
/user1/egon/kshueb
$ cd;pwd
/user1/egon
$ cd -
/user1/egon/kshueb
$ cd /usr/include
$ cd include news
/usr/news
$ cd;pwd
/user1/egon
$ cd egon email
/user1/email
$ cd -
/user1/egon
$ cd kshueb
/user1/egon/kshueb
$
```

**Hinweis** Das Kommando **cd** kann nicht ausgeführt werden, wenn die Option **restricted**<sup>1</sup> gesetzt ist.

1. Siehe Kapitel 5.26.

## 5.24.2 Lokalisieren bzw. Klassifizieren eines Kommandos (**whence**)

Das builtin-Kommando **whence** ist dem builtin-Kommando **type** der Bourne-Shell ähnlich. **whence** ermöglicht es, den absoluten Pfadnamen oder den Typ eines Kommandos zu ermitteln. Die Aufrufsyntax für **whence** ist:

**whence** [-v] *name(n)*

Ist die Option **-v** nicht angegeben, so wird zu allen angegebenen *name(n)* deren absoluter Pfadname ausgegeben, wenn ein solcher existiert.

Wird die Option **-v** angegeben, so wird zu jedem angegebenen *name(n)* dessen Typ ausgegeben:

- ▶ Schlüsselwort (*keyword*)
- ▶ Alias (*alias*)
- ▶ Exportiertes Alias (*exported alias*)
- ▶ Builtin-Kommando (*builtin*)
- ▶ undefinierte Funktion (*undefined function*)
- ▶ Funktion (*function*)
- ▶ Mit **-t** markiertes Alias (*tracked alias*)
- ▶ Programm (*program*)
- ▶ nicht gefunden (*not found*)

**type** ist im übrigen ein vordefiniertes Alias<sup>1</sup> der Form

**type='whence -v'**

Der exit-Status von **whence** ist 0, wenn alle angegebenen *name(n)* wirklich existieren, ansonsten verschieden von 0.

### Beispiel

```
$ whence ed ↵
/bin/ed
$ whence -v ed ↵
ed is a tracked alias for /bin/ed
$ whence echo ↵
echo
$ whence -v echo ↵
echo is a shell builtin
$ type who funk pwd ↵
who is a tracked alias for /bin/who
funk not found
pwd is a shell builtin
$ type functions ↵
```

---

1. Siehe Kapitel 5.17.5.

```
functions is an exported alias for typeset -f
$ whence cd repeat
cd
function _repeat {
$ whence -v cd repeat
cd is a shell builtin
repeat is an alias for function _repeat {
$ type isprim
isprim is an undefined function
$ type while
while is a keyword
$ type type
type is an exported alias for whence -v
$
```

### 5.24.3 Setzen von Optionen für die aktuelle ksh (set)

Es ist möglich, während des Arbeitens mit der **ksh**, neue Optionen ein- bzw. auszuschalten oder die Positionsparameter neu zu setzen bzw. zu löschen. Dazu steht das builtin-Kommando **set** zur Verfügung. Die Aufrufsyntax für **set** ist:

**set** [**±aefhkmnopstuvx-**] [**±o optname**]... [**±A name**] [**argument(e)**]<sup>1</sup>

#### Beschreibung

Das Kommando **set** kann recht vielseitig verwendet werden:

- ▶ Zum Einschalten von Optionen: **-option** oder **-o optname**
- ▶ Zum Ausschalten von Optionen: **+option** oder **+o optname**
- ▶ Zum expliziten Setzen der Positionsparameter mit der Angabe von **argument(e)**
- ▶ Zum Zuweisen von Werten an eine Array-Variable. Dazu muß **name** und **argument(e)** angegeben sein.
- ▶ Zum Sortieren der Positionsparameter oder der angegebenen **argument(e)**. Dazu muß die Option **-s** verwendet werden.
- ▶ Zum Löschen der Positionsparameter. Dazu muß **set --** aufgerufen werden.
- ▶ Zum Auflisten der Werte und Namen aller momentan definierten Shell-Variablen. Dazu muß **set** alleine (ohne Angabe von **optionen** oder **argument(e)**) aufgerufen werden.

---

1. [**±optname**] kann bei einem **set**-Aufruf mehrmals angegeben werden.

- Zum Auflisten des momentanen Zustands aller Optionen (ein- oder ausgeschaltet). Dazu muß **set -o** aufgerufen werden.

## Optionen

**set** kennt folgende Optionen:

| Optionen          | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-a</b>         | <b>(allexport)</b> markiert die Shell-Variablen, die verändert oder neu angelegt werden, für den Export.                                                                                                                                                                                                                                                                                                                                                                       |
| <b>-e</b>         | <b>(errexit)</b> wenn ein Kommando in einem Shell-Skript nicht erfolgreich ausgeführt werden kann, dann wird zunächst ein eventuell zu <b>ERR</b> installierter Signalhandler ausgeführt und danach sofort die Ausführung des Shell-Skripts beendet.<br><br>Während des Lesens von <i>.profile</i> oder der Environment-Datei wird diese Option von der <b>ksh</b> ignoriert.                                                                                                  |
| <b>-f</b>         | <b>(noglob)</b> schaltet die Dateinamen-Expandierung aus: Metazeichen wie <b>* ? [</b> werden als einfache Zeichen ohne Sonderbedeutung behandelt.                                                                                                                                                                                                                                                                                                                             |
| <b>-h</b>         | <b>(trackall)</b> bewirkt, daß für jedes aufgerufene Kommando automatisch ein sogenanntes Tracked Alias mit dem gleichen Namen wie das betreffende Kommando definiert wird. Diesem Alias wird dann als Wert der Pfadname des entsprechenden Kommandos zugewiesen, um bei einem weiteren Aufruf ein erneutes Durchsuchen der <b>PATH</b> -Directories nach diesen Kommandonamen zu vermeiden.<br><br>Diese Option wird bei nicht interaktiven Shells automatisch eingeschaltet. |
| <b>-k</b>         | <b>(keyword)</b> bewirkt, daß <u>alle</u> Shell-Variablen an die Umgebung ( <i>environment</i> ) eines Kommandos übergeben werden. Normalerweise werden Variablen-Zuweisungen nur dann an aufgerufene Kommandos übergeben, wenn sie beim Aufruf vor dem Kommandonamen angegeben sind <sup>a</sup> .                                                                                                                                                                            |
| <b>-m</b>         | <b>(monitor)</b> Hintergrund-Jobs werden in einer eigenen Prozeß-Gruppe ausgeführt. Die Beendigung des jeweiligen Hintergrund-Jobs wird dabei immer mit dessen exit-Status gemeldet.<br><br>Diese Option wird auf Systemen, die über Job-Kontrolle verfügen, automatisch für interaktive Shells eingeschaltet.                                                                                                                                                                 |
| <b>-n</b>         | <b>(noexec)</b> Kommandos aus einem Shell-Skript werden nur gelesen und auf Syntaxfehler untersucht, aber nicht ausgeführt.<br><br>Eine interaktive <b>ksh</b> ignoriert diese Option.                                                                                                                                                                                                                                                                                         |
| <b>-o optname</b> | Für <i>optname</i> kann dabei folgendes angegeben werden:<br><br><b>allexport</b> entspricht <b>-a</b>                                                                                                                                                                                                                                                                                                                                                                         |

| Optionen          | Bedeutung                                                                                                                                                                                                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>bgnice</b>     | alle Hintergrund-Jobs werden mit einer niedrigeren Priorität ausgeführt                                                                                                                                                                                                                             |
| <b>errexist</b>   | entspricht <b>-e</b>                                                                                                                                                                                                                                                                                |
| <b>emacs</b>      | schaltet builtin-Editor <b>emacs</b> ein                                                                                                                                                                                                                                                            |
| <b>gmacs</b>      | schaltet builtin-Editor <b>gmacs</b> ein                                                                                                                                                                                                                                                            |
| <b>ignoreeof</b>  | Eine interaktive <b>ksh</b> kann nur mit <b>exit</b> (nicht mit <i>Strg-d</i> ) verlassen werden.                                                                                                                                                                                                   |
| <b>keyword</b>    | entspricht <b>-k</b>                                                                                                                                                                                                                                                                                |
| <b>markdirs</b>   | An alle Directory-Namen, die aus Dateinamen-Expandierung resultieren, wird ein Slash / angehängt.                                                                                                                                                                                                   |
| <b>monitor</b>    | entspricht <b>-m</b>                                                                                                                                                                                                                                                                                |
| <b>noclobber</b>  | Bei Angabe des Umlenkungsoperators > werden existierende Dateien nicht überschrieben. Ein Überschreiben von existierenden Dateien ist in diesem Fall nur mit dem Umlenkungsoperator >  möglich. Diese Option ist nur auf <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden. |
| <b>noexec</b>     | entspricht <b>-n</b>                                                                                                                                                                                                                                                                                |
| <b>noglob</b>     | entspricht <b>-f</b>                                                                                                                                                                                                                                                                                |
| <b>nolog</b>      | Funktionsdefinitionen werden nicht in der History-Datei gespeichert.<br><br>Diese Option ist nur auf <b>ksh</b> -Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.                                                                                                                     |
| <b>nounset</b>    | entspricht <b>-u</b>                                                                                                                                                                                                                                                                                |
| <b>privileged</b> | entspricht <b>-p</b> ( <b>ksh</b> -Versionen nach 3.6.1986)                                                                                                                                                                                                                                         |
| <b>protected</b>  | entspricht <b>-p</b> ( <b>ksh</b> -Version vom 3.6.1986)                                                                                                                                                                                                                                            |
| <b>trackall</b>   | entspricht <b>-h</b>                                                                                                                                                                                                                                                                                |
| <b>verbose</b>    | entspricht <b>-v</b>                                                                                                                                                                                                                                                                                |
| <b>vi</b>         | schaltet den builtin-Editor <b>vi</b> ein                                                                                                                                                                                                                                                           |
| <b>viraw</b>      | Jedes eingegebene Zeichen wird so interpretiert, als ob es im <b>vi</b> -Modus eingegeben wurde.                                                                                                                                                                                                    |
| <b>xtrace</b>     | entspricht <b>-x</b>                                                                                                                                                                                                                                                                                |



| Optionen  | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-p</b> | <p>Diese Option, welche für Systemadministratoren von Interesse sein kann, hat unterschiedliche Bedeutungen auf den einzelnen <b>ksh</b>-Versionen:</p> <p><b>privileged</b> (<b>ksh</b>-Versionen nach 3.6.1986)</p> <p>Wenn ausgeschaltet, so wird die effektive UID auf die reale UID und die effektive GID auf die reale GID gesetzt.</p> <p>Wenn eingeschaltet, dann werden die effektive UID und GID wieder auf die Werte gesetzt, die vorlagen, als die <b>ksh</b> aufgerufen wurde.</p> <p><b>privileged</b> ist immer dann eingeschaltet, wenn die effektive UID nicht gleich der realen UID oder die effektive GID nicht gleich der realen GID ist.</p> <p>Wenn <b>privileged</b> eingeschaltet ist, dann gilt:</p> <ul style="list-style-type: none"> <li>– <b>\$HOME/.profile</b> kann nicht ausgeführt werden.</li> <li>– Anstelle der über <b>ENV</b> festgelegten Environment-Datei wird die Datei <b>/etc/suid_profile</b> gelesen. Der Systemadministrator kann diese Datei benutzen, um beispielsweise die <b>PATH</b>-Variable zurückzusetzen.</li> </ul> <p><b>protected</b> (<b>ksh</b>-Version vom 3.6.1986)</p> <p>Wird automatisch eingeschaltet, wenn die effektive UID nicht der realen UID oder die effektive GID nicht der realen GID ist.</p> <p>Wenn <b>protected</b> eingeschaltet ist, dann gilt:</p> <ul style="list-style-type: none"> <li>– <b>\$HOME/.profile</b> kann nicht ausgeführt werden.</li> <li>– die <b>PATH</b>-Variable wird auf ihren default-Wert zurückgesetzt.</li> <li>– Anstelle der über <b>ENV</b> festgelegten Environment-Datei wird die Datei <b>/etc/suid_profile</b> gelesen. Der Systemadministrator kann diese Datei benutzen, um beispielsweise die <b>PATH</b>-Variable zurückzusetzen.</li> </ul> |
| <b>-s</b> | bewirkt, daß die Positionsparameter bzw. die angegebenen <i>argument(e)</i> sortiert werden. Beim Aufruf einer <b>ksh</b> hat diese Option jedoch eine andere Bedeutung <sup>b</sup> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>-t</b> | Nach dem Lesen und der Ausführung eines Kommandos wird die entsprechende <b>ksh</b> beendet.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>-u</b> | ( <b>nounset</b> ) Ein Zugriff auf nicht gesetzte Shell-Variable wird als Fehler gewertet. Ohne diese Option setzt die Shell üblicherweise die leere Zeichenkette für eine solche Variable ein.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>-v</b> | ( <b>verbose</b> ) bewirkt, daß alle Shell-Eingabezeilen – so wie sie gelesen werden – ausgegeben werden. Erst nach dieser Ausgabe wird das entsprechende Kommando ausgeführt.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

| Optionen  | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-x</b> | ( <b>xtrace</b> ) bewirkt, daß jedes einfache Kommando unmittelbar vor seiner Ausführung (nach Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung) auf der Standardfehlerausgabe angezeigt wird. Vor dieser Kommandozeile wird dabei noch der Inhalt der Shellvariablen <b>PS4</b> <sup>f</sup> angegeben. Erst nach dieser Ausgabe wird die betreffende Kommandozeile ausgeführt. |
| <b>-</b>  | (Minuszeichen) schaltet zunächst die beiden Optionen <b>-x</b> und <b>-v</b> aus und behandelt den Rest der Kommandozeile als normale Argumente, selbst wenn diese mit <b>-</b> (Minus) beginnen.                                                                                                                                                                                                       |
| <b>--</b> | (doppeltes Minuszeichen) hat keine Auswirkung auf die Optionen. Wenn nach <b>--</b> keine weiteren Argumente angegeben sind, so werden die Positionsparameter gelöscht.                                                                                                                                                                                                                                 |

- Diese Option wurde nur aus Gründen der Kompatibilität zur Bourne-Shell in die **ksh** übernommen. Eventuell wird diese Option in späteren **ksh**-Versionen auch wieder entfernt.
- Siehe Kapitel 5.26.2.
- Die Variable **PS4** (siehe Kapitel 5.8.2) ist nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden. Frühere **ksh**-Versionen geben anstelle von **PS4** das Zeichen **+** (wie die Bourne-Shell) aus.

Allgemein gilt, daß die Optionenangabe immer dann als abgeschlossen gilt, wenn ein Argument, das nicht mit **+** oder **-** beginnt, oder ein einfaches bzw. doppeltes Minuszeichen vorkommt.

Folgen also nach einem **-** oder **--** weitere Argumente, so werden diese immer als normale Argumente gewertet, selbst wenn sie mit einem **-** (Minus) beginnen. Um also den Positionsparameter **\$1** mit einem String zu besetzen, der mit **-** (Minus) beginnt, muß **-** bzw. **--** verwendet werden, z.B.

```
$ set -- -a
$ echo $1
-a
$
```

Automatisch einzuschaltende Optionen sollten üblicherweise in **\$HOME/.profile** oder in der Environment-Datei gesetzt werden<sup>1</sup>.

Der automatische Parameter **\$-** enthält die gesetzten Optionen für die momentan aktive Shell. Mit dem Aufruf

**echo \$-**

können diese gerade gesetzten Optionen ausgegeben werden.

1. Siehe Kapitel 5.27.

### Die Angabe

**±A name**<sup>1</sup>

muß benutzt werden, wenn einem Array *name* die *argument(e)* als Werte zugewiesen werden sollen. Falls zusätzlich noch die Option **-s** angegeben ist, werden diese Werte vor der Zuweisung noch sortiert.

Soll der alte Inhalt des Arrays *name* vor der Zuweisung zuerst gelöscht werden, muß

**-A name**

und ansonsten

**+A name**

angegeben werden.

Der exit-Status des **set**-Kommandos ist immer 0 (erfolgreich).

#### Beispiel

```
$ set -o emacs -o trackall ↵
```

```
$ set -o ↵
```

```
Current option settings
```

```
allexport off
```

```
bgnice on
```

```
emacs on
```

```
errexit off
```

```
gmacs off
```

```
ignoreeof off
```

```
interactive on
```

```
keyword off
```

```
markdirs off
```

```
monitor on
```

```
noexec off
```

```
noclobber off
```

```
noglob off
```

```
nolog off
```

```
nounset off
```

```
privileged off
```

```
restricted off
```

```
trackall on
```

```
verbose off
```

```
vi off
```

```
viraw off
```

```
xtrace off
```

```
$ set -o | grep ignore ↵
```

```
ignoreeof off
```

```
$ set eins zwei drei ↵
```

```
$ echo $* ↵
```

```
eins zwei drei
```

---

1. Nur auf **ksh**-Versionen verfügbar, die nach dem 3.6.1986 freigegeben wurden.

```
$ set -- [Alle Positionsparameter löschen]
$ echo $*

$ echo $-
ismh
$ set -o noglob -o ignoreeof
$ echo $-
isfmh
$ set
:
ENV=/user1/egon/.ksh_env
ERRNO=0
FCEDIT=/bin/ed
FPATH=/user1/egon/flib
HOME=/user1/egon
HZ=60
IFS=

LINENO=1
LOGNAME=egon
MAIL=/usr/spool/mail/egon
MAILCHECK=600
OLDPWD=/user1/egon
OPTIND=0
PATH=/bin:/usr/bin:.
PPID=1
PS1=!>$PWD:
PS2=>
PS3=#?
PS4=Zeile $LINENO:
PWD=/user1/egon/kshueb
RANDOM=30326
SECONDS=5088
SHELL=/bin/ksh
TERM=ansi
TMOUT=0
TZ=MET-1MESZ,M3.5.0,M9.5.0/03
_=set
from=; _from
$ set -s mueller zeppelin aachen berta
$ echo $*
aachen berta mueller zeppelin
$ ls a*
a*
$ set +o noglob
$ ls a*
a*
abc
abc2
add
addiere.c
```

```

ausgab
$ set +A zahl null eins zwei drei
$ echo ${zahl[2]}
zwei
$ set +A zahl zero one
$ echo ${zahl[0]} ${zahl[2]}
zero zwei
$ set -A zahl zero one
$ echo ${zahl[0]} ${zahl[2]}
zero
$ set eins zwei gsuffa
$ a=""
$ set $a [resultiert in einem set-Aufruf ohne Argumente]
: [Ausgabe aller momentan definierten Shell-Variablen
: und -Funktionen]
:
$ echo $1
eins
$ set - $a
$ echo $1
eins
$

```

Um für die Argumente eines Funktionsaufrufs die Dateinamen-Expanderung auszuschalten, ist folgende Vorgehensweise empfehlenswert:

```

alias funktionsname='set -o noglob; _funktionsname'
function _funktionsname
{
 trap 'set +o noglob' EXIT

}

```

**Beispiel**

Die nachfolgenden Deklarationen und Definitionen könnten in der Environment-Datei (\$ENV) eingetragen werden:

```

$ cat $ENV
:
:
typeset -fx _pot
alias pot='set -o noglob; _pot'

function _pot
{
 trap 'set +o noglob' EXIT
 if [[$# -ne 2]]
 then echo "usage: $0 operand potenz" >&2
 return 1
 fi
}

```

```
integer op=$1 pot=$2 ergeb=1 i=1
if isnum $op && isnum $pot
then
 while ((i<=pot))
 do ergeb=ergeb*op
 i=i+1
 done
 print $ergeb
else
 echo "Nur Integer-Zahlen erlaubt" >&2
 return 1
fi
}
$. $ENV [Neue Definitionen aus Env.-Datei der ksh bekannt machen]
$ pot 2 10
1024
$ ls a*c
abc
addiere.c
$ a=5 c=4
$ _pot a*c 2 [Bei _pot wird Dateinamen-Exp. nicht ausgeschaltet]
usage: _pot operand potenz
$ pot a*c 2
400
$ pot 123456 0
1
$ pot 5 -3
Nur Integer-Zahlen erlaubt
$
```

#### 5.24.4 Die builtin-Kommandos times und ulimit

Die beiden letzten noch nicht besprochenen builtin-Kommandos, die sich von der Bourne-Shell unterscheiden, sind **times** und **ulimit**:

##### **times**

Das Kommando **times**<sup>1</sup> gibt die bisher verbrauchte CPU-Zeit der momentan aktiven **ksh** aus; zusätzlich gibt dieses **times** noch die verbrauchte CPU-Zeit aller Kindprozesse aus, die von dieser **ksh** gestartet wurden.

Es werden dabei zwei Zeilen auf die Standardausgabe geschrieben:

1. Zeile: **ksh**-Zeiten
2. Zeile: Zeiten der Kindprozesse

---

1. **times** verfügt über keinerlei Argumente.

Pro Zeile werden dabei zwei Zeiten ausgegeben:

1. Zeit(user) gebrauchte CPU-Zeit im Benutzermodus
2. Zeit(sys) gebrauchte CPU-Zeit im Systemmodus (z.B. bei der Ausführung von Systemroutinen)

Der exit-Status des Kommandos **times** ist immer 0 (erfolgreich).

**Beispiel**

```
$ times ↵
0m3.50s 0m2.18s
0m1.06s 0m2.18s
$ ksh ↵
$ times ↵
0m0.41s 0m0.26s
0m0.00s 0m0.00s
$ exit ↵
$ times ↵
0m3.53s 0m2.25s
0m1.48s 0m2.46s
$ { find / -name "*.c" -print 2>&- | wc -l; times; } ↵
 1263
0m3.55s 0m2.30s
0m3.65s 0m19.65s
$
```

**ulimit**

Das builtin-Kommando **ulimit** der **ksh** unterscheidet sich von dem der Bourne-Shell dadurch, daß es die Angabe von Optionen zuläßt.

Mit **ulimit** ist es möglich, Limits für Systemressourcen (wie z.B. Dateien, Daten-segment, Stacksegment oder CPU-Zeit) festzulegen oder sich die momentan gesetzten Limits dafür anzeigen zu lassen. Die entsprechenden Limits gelten dabei für die aktuelle **ksh** und für jeden Kindprozeß, der von dieser Shell gestartet wird.

Die Aufrufsyntax für **ulimit** ist:

**ulimit** [-acdfmpst] [*n*]<sup>1</sup>

Ist *n* angegeben, so wird damit ein Limit für die entsprechende Systemressource festgelegt. Für *n* kann ein beliebiger arithmetischer Ausdruck oder **unlimited** (unbegrenzt) angegeben werden.

Ist *n* nicht angegeben, so wird das momentan festgelegte Limit für die entsprechende Systemressource ausgegeben.

---

1. Die hier angegebenen Optionen werden nicht von allen **ksh**-Versionen angeboten.

Wie groß maximal ein Limit-Wert festgesetzt werden kann, ist von System zu System verschieden. Zudem ist es auch nicht auf jedem System möglich, alle nachfolgend angegebenen Optionen zu benutzen:

| Option    | Bedeutung                                                                                                                                                           |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-a</b> | ( <b>all</b> ) Alle momentan gültigen Limits anzeigen                                                                                                               |
| <b>-c</b> | ( <b>core</b> ) Limit für sogenannte <i>core dumps</i> auf <i>n</i> Blöcke festlegen                                                                                |
| <b>-d</b> | ( <b>data segment</b> ) Limit für das Datensegment auf <i>n</i> Kilobytes festlegen                                                                                 |
| <b>-f</b> | ( <b>file</b> ) Limit für Dateien, die beschrieben werden, auf <i>n</i> Blöcke festlegen <sup>a</sup>                                                               |
| <b>-m</b> | ( <b>memory</b> ) Limit für den physikalischen Hauptspeicher, der von diesem Prozeß oder seinen Kindprozessen benutzt werden darf, auf <i>n</i> Kilobytes festlegen |
| <b>-p</b> | ( <b>pipe</b> ) Limit für eine Pipe auf <i>n</i> Bytes festlegen                                                                                                    |
| <b>-s</b> | ( <b>stack segment</b> ) Limit für das Stacksegment auf <i>n</i> Kilobytes festlegen                                                                                |
| <b>-t</b> | ( <b>time</b> ) Limit für die CPU-Zeit auf <i>n</i> Sekunden festlegen                                                                                              |

a. Für zu lesenden Dateien gibt es keine Größenbegrenzung.

Falls keine Option angegeben ist, so entspricht dies der Angabe der Option **-f**.

Der exit-Status des Kommandos **ulimit** ist immer 0 (erfolgreich).

**Beispiel**

```
$ ulimit -f 500
$ ulimit -f
500
$ ulimit
500
$
```

## 5.24.5 Zusammenfassung der builtin-Kommandos

Hier werden nun alle builtin-Kommandos der **ksh** nochmals zusammengefaßt. Dabei wird zum Zwecke der Wiederholung eine kurze Beschreibung gegeben. Eine genauere Beschreibung der jeweiligen builtin-Kommandos kann im beigelegten Referenzheftchen nachgeschlagen werden, wo zu allen hier behandelten Kommandos eine alphabetisch geordnete Liste angegeben ist.



Die Korn-Shell verfügt über folgende builtin-Kommandos:

| Kommando                                                           | Wirkung                                                                                                                     |
|--------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>#</b>                                                           | Kommentar-Kommando                                                                                                          |
| <b>:</b> <i>[argument(e)]</i>                                      | Null-Kommando                                                                                                               |
| <b>.</b> <i>kdodatei [argument(e)]</i>                             | Punkt-Kommando:<br><br>Liest die Kommandos in <i>kdodatei</i> und führt sie in der aktuellen Shell aus.                     |
| <b>alias</b> <b>[-tx]</b> <i>[name[=wert] ...]</i>                 | Definieren bzw. Anzeigen von Aliase                                                                                         |
| <b>bg</b> <i>[job(s)]</i>                                          | Ausführung von angehaltenen <i>job(s)</i> im Hintergrund fortsetzen                                                         |
| <b>break</b> <i>[n]</i>                                            | Verlassen einer bzw. von <i>n</i> umschließenden <b>for</b> -, <b>while</b> -, <b>until</b> - oder <b>select</b> -Schleifen |
| <b>cd</b> <i>[directory]</i>                                       | Wechseln in ein anderes Working-Directory                                                                                   |
| <b>cd</b> <i>altstring neustring</i>                               | Wechseln in ein anderes Working-Directory                                                                                   |
| <b>continue</b> <i>[n]</i>                                         | Abbrechen eines bzw. von <i>n</i> <b>for</b> -, <b>while</b> -, <b>until</b> - oder <b>select</b> -Schleifendurchläufen     |
| <b>echo</b> <b>[-n]</b> <i>[argument(e)]</i>                       | Ausgeben der <i>argument(e)</i> auf Standardausgabe                                                                         |
| <b>eval</b> <i>[argument(e)]</i>                                   | Ausführen der <i>argument(e)</i> als Kommandos                                                                              |
| <b>exec</b> <i>[argument(e)]</i>                                   | Überlagern der Shell mit einem Kommando                                                                                     |
| <b>exit</b> <i>[n]</i>                                             | Beenden der momentan aktiven Shell                                                                                          |
| <b>export</b> <i>[variable[=wert]] ....</i>                        | Exportieren von Shell-Variablen                                                                                             |
| <b>fc</b> <b>-e</b> <b>-</b> <i>[alt=neu]</i><br><i>[kommando]</i> | Erneutes Ausführen zuvor eingegebener Kommandozeilen                                                                        |
| <b>fc</b> <b>[-e editor]</b> <b>[-nlr]</b><br><i>[von [bis]]</i>   | Auflisten zuvor eingegebener Kommandozeilen                                                                                 |
| <b>fg</b> <i>[job(s)]</i>                                          | Ausführung von Hintergrund- <i>job(s)</i> im Vordergrund fortsetzen                                                         |
| <b>getopts</b> <i>optstring name</i><br><i>[argument(e)]</i>       | Auswerten der Optionen in einem Shell-Skript                                                                                |
| <b>jobs</b> <b>[-lp]</b> <i>[job(s)]</i>                           | Ausgeben von Informationen zu <i>job(s)</i>                                                                                 |
| <b>kill</b> <b>[-signal]</b> <i>job(s)</i>                         | Schicken von Signalen an Jobs                                                                                               |
| <b>kill</b> <b>-l</b>                                              | Auflisten der Signalnamen mit Nummern                                                                                       |
| <b>let</b> <i>argument(e)</i>                                      | Auswerten von arithmetischen Ausdrücken                                                                                     |
| <b>newgrp</b> <b>[-]</b> <i>[gruppenname]</i>                      | Wechseln der Gruppenzugehörigkeit                                                                                           |
| <b>print</b> <b>[-Rnprs]</b> <i>[n]</i><br><i>[argument(e)]</i>    | Ausgeben auf die Standardausgabe                                                                                            |

| Kommando                                                                     | Wirkung                                                                                                                                                                                                                                                                                                         |
|------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pwd</b>                                                                   | Ausgeben des Working-Directorys                                                                                                                                                                                                                                                                                 |
| <b>read</b> [-prsu[n]] [variable(n)]                                         | Lesen einer Zeile von der Standardeingabe                                                                                                                                                                                                                                                                       |
| <b>readonly</b> [variable[=wert]]<br>....                                    | Kennzeichnen von Shell-Variablen als "nur lesbar"                                                                                                                                                                                                                                                               |
| <b>return</b> [n]                                                            | Verlassen einer Funktion                                                                                                                                                                                                                                                                                        |
| <b>set</b> [±aefhkmnopstuvx-]<br>[±o optname]... [ ±A name]<br>[argument(e)] | Ein- bzw. Ausschalten von <b>ksh</b> -Optionen oder Setzen bzw. Löschen von Positionsparametern                                                                                                                                                                                                                 |
| <b>shift</b> [n]                                                             | Verschieben der Werte von Positionsparametern                                                                                                                                                                                                                                                                   |
| <b>test</b> [ausdr]                                                          | Auswerten des Ausdrucks <i>ausdr</i> :<br><br>Anstelle von<br><b>test</b> <i>ausdr</i> kann auch [ <i>ausdr</i> ] angegeben werden. <b>ksh</b> -Versionen, die nach dem 3.6.1986 freigegeben wurden, bieten das Kommando <b>[...]</b> an und machen somit das Kommando <b>test</b> bzw. <b>[.]</b> überflüssig. |
| <b>times</b>                                                                 | Anzeigen der bisher von der aktuellen Shell verbrauchten CPU-Zeit (Benutzer- und Systemzeit).                                                                                                                                                                                                                   |
| <b>trap</b> [argument] [signalnummer(n)]                                     | Installieren bzw. Auflisten von Signalhandlern                                                                                                                                                                                                                                                                  |
| <b>typeset</b> ±f [tux] [name(n)]                                            | Ein- / Ausschalten bzw. Anzeigen von Attributen für Funktionen                                                                                                                                                                                                                                                  |
| <b>typeset</b> [±HLRZilr-tux[n]] [variable[=wert]] ...                       | Ein- / Ausschalten bzw. Anzeigen von Attributen für Variablen                                                                                                                                                                                                                                                   |
| <b>ulimit</b> [-acdfmpst] [n]                                                | Festlegen bzw. Auflisten von Limits für Systemressourcen                                                                                                                                                                                                                                                        |
| <b>umask</b> [3-stellige-oktalzahl]                                          | Setzen bzw. Ausgeben der Dateikreierungs-Maske                                                                                                                                                                                                                                                                  |
| <b>unalias</b> name(n)                                                       | Löschen der Aliase <i>name(n)</i>                                                                                                                                                                                                                                                                               |
| <b>unset</b> [-f] name(n)                                                    | Löschen von Shell-Variablen oder Shell-Funktionen                                                                                                                                                                                                                                                               |
| <b>wait</b> [job(s)]                                                         | Warten auf die Beendigung von Jobs                                                                                                                                                                                                                                                                              |
| <b>whence</b> [-v] name(n)                                                   | Lokalisieren bzw. Klassifizieren eines Kommandos                                                                                                                                                                                                                                                                |

### Vordefinierte Aliase

Die von der **ksh** vordefinierten Aliase sind eigentlich keine builtin-Kommandos sondern fast alle (bis auf **nohup**) nur Aliase für bestimmte Aufrufformen von builtin-Kommandos. Deshalb werden sie hier nochmals alle zusammengefaßt:

```
autoload (typeset -fu)
false (let 0)
functions (typeset -f)
hash (alias -t)
history (fc -l)
integer (typeset -i)
nohup (nohup)
r (fc -e -)
true (:)
type (whence -v)
```

Auf manchen **ksh**-Versionen sind auch noch die folgenden Aliase bereits vordefiniert:

```
stop (kill -STOP)
suspend (kill -STOP $$)
```

## 5.25 Die Abarbeitung von Kommandozeilen

Die Abarbeitung von Kommandozeilen ist für die **ksh** wesentlich genauer festgelegt als für die Bourne-Shell. Allerdings gilt auch hier die Einschränkung, daß es sich nur um einen Vorschlag der beiden Erfinder der Korn-Shell handelt, und sich nicht jedes System zwingend an diese Reihenfolge halten muß. Nach dieser Vorgabe werden Kommandozeilen von der **ksh** in drei Schritten abgearbeitet.

Im ersten Schritt liest die **ksh** die Kommandozeile und zerteilt diese in einzelne Token. Sie bestimmt dann, ob es sich dabei um ein einfaches Kommando handelt oder nicht. So kann sie feststellen, wieviel sie lesen muß.

Im zweiten Schritt expandiert die **ksh** die Kommandozeile, im dritten Schritt führt sie diese aus. Programmiersprach-Konstrukte wie die **while**-Schleife werden dabei bei jedem Schleifendurchlauf neu expandiert und ausgeführt.

## 1. Schritt: Lesen von Kommandos

### 1.1 Zerlegen der Kommandozeile in einzelne Token

Ein Token ist dabei:

- ▶ Ein-/Ausgabeumlenkungs-Operator
- ▶ Neuezeile-Zeichen
- ▶ Reserviertes Wort:

```
if then else elif fi
case esac
for while until do done
{ }
select in
function
time
[[]]¹
```

- ▶ Bezeichner
- ▶ Wort
- ▶ Hier-Dokument

Wenn die **ksh** erkennen kann, daß ein Token noch unvollständig ist, und es wurde bereits  gedrückt, dann gibt die **ksh** den Sekundär-Promptstring aus **PS2** aus, um anzuzeigen, daß das Token noch unvollständig ist:

```
$ print "Hallo" 
> egon" 
Hallo
egon
$
```

Während des Zerteilens einer Kommandozeile in einzelne Token führt die **ksh** Alias-Substitution aus (siehe Schritt 1.4):

```
$ alias text='Hallo \'' 
$ alias d="print " 
$ d text 
> egon 
Hallo egon
$
```

---

1. Nur auf **ksh**-Versionen nach dem 3.6.1986 verfügbar.

## 1.2 Festlegen des Kommandotyps

Wenn das erste Token eines der folgenden reservierten Wörter ist, dann muß die **ksh** weitere Zeilen lesen, bis sie das Ende dieses Kommandos gefunden hat:

```
{ case for function if until select time while ((([[
```

Wenn das erste Token einer Kommandozeile eines der folgenden Wörter ist, und die **ksh** ist nicht interaktiv, gibt sie eine Fehlermeldung aus und beendet sich:

- ▶ eines der reservierten Wörter  
`do done elif else esac fi in then } ]]`
- ▶ eines der Konstrukte `| | & && ; ; | &`

Jedes andere Token am Anfang einer Zeile legt fest, daß es sich um ein einfaches Kommando handelt.

## 1.3 Lesen von einfachen Kommandos

Die **ksh** liest alle Token bis zu einem der folgenden Konstrukte als einfache Kommandos:

`; | & | | && | &` *Neuzeile-Zeichen*

Die **ksh** teilt dabei die gelesenen Token in drei Klassen ein:

- ▶ Ein- / Ausgabeumlenkungs-Anweisungen
- ▶ Variablen-Zuweisungen<sup>1</sup>
- ▶ Kommando-Wörter (Kommandoname und die Kommando-Argumente)  
Die **ksh** überprüft dabei, ob das erste Kommando-Wort ein Alias ist (siehe Schritt 1.4)

## 1.4 Alias-Substitution

Für jedes einfache Kommando prüft die **ksh** das erste Wort (Kommandoname), um dieses eventuell als Alias zu klassifizieren. Wenn keinerlei Quoting in diesem Wort vorkommt und das Wort ein erlaubter Alias-Name ist, der nicht mit **-t** markiert wurde, dann setzt die **ksh** für den Alias, falls definiert, den entsprechenden Alias-Text ein. Falls im Alias-Text der Alias-Name selbst nochmals vorkommt, wird dieser nicht erneut substituiert. Kommt im Alias-Text dagegen ein anderer Alias-Name vor, wird auf **ksh**-Versionen nach dem 3.6.1986 auch dieser substituiert.

---

1. Falls die Option **keyword** ausgeschaltet ist, werden nach dem Kommandonamen angegebene Zuweisungen nicht als solche, sondern als Kommando-Wörter klassifiziert; siehe Kapitel 5.24.3.

Normalerweise wird die Alias-Substitution nur für das erste Wort eines einfachen Kommandos durchgeführt. Endet jedoch ein Alias-Text mit einem Leer- oder Tabulatorzeichen, wird auch noch für das nachfolgende Wort Alias-Substitution durchgeführt.

## 1.5 Tilde-Expandierung

Nachdem die **ksh** die Alias-Substitution durchgeführt hat, prüft sie jedes Wort in der Kommandozeile, ob es eventuell mit `~` (engl. *tilde*) beginnt. Ist ein solches Wort vorhanden, führt sie für dieses Wort vom Anfang bis zum nächsten Vorkommen von `/` bzw. bis zum Ende die sogenannte *Tilde-Expandierung* durch:

| Text                       | Tilde-Expandierung (wird ersetzt durch)     |
|----------------------------|---------------------------------------------|
| <code>~</code>             | <b>\$HOME</b>                               |
| <code>~+</code>            | <b>\$PWD</b>                                |
| <code>~-</code>            | <b>\$OLDPWD</b>                             |
| <code>~loginname</code>    | <b>\$HOME</b> vom Benutzer <i>loginname</i> |
| <code>~anderer-text</code> | keine Ersetzung                             |

### Hinweis

Bei Variablenzuweisungen führt die **ksh** Tilde-Expandierung durch, wenn `~` am Anfang des entsprechenden Werts (unmittelbar nach dem Gleichheitszeichen) angegeben ist. **ksh**-Versionen vom 3.6.1986 und früher führten Tilde-Expandierungen sogar nach jedem `:` (Doppelpunkt) in einem Wert einer Variablenzuweisung durch.

### Beispiel

```
$ pwd
/user1/egon/kshueb
$ alias d='print '
$ alias e=emil
$ d e
/user1/emil
$
```

## 2. Schritt: Expandieren der Kommandozeile

Vor der Ausführung untersucht die **ksh** jedes Token, um daraus die endgültige Kommandozeile zu formen. Die **ksh** führt dabei von links nach rechts Kommando- und Parametersubstitution für jedes entsprechende Wort durch. Danach zerteilt sie die so entstandene Kommandozeile in einzelne Wörter, für welche dann noch eventuell Dateinamen-Expandierung durchgeführt wird. Das Entfernen von Quoting-Zeichen geschieht in diesem Schritt zuletzt.

Die nachfolgende Tabelle zeigt, was für die drei einzelnen Klassen von Tokens, die in Schritt 1.3 vorgestellt wurden, von der **ksh** durchgeführt wird:

|                                   | <i>Ein-/Ausg.-<br/>Umlenkung</i> | <i>Variablen-<br/>Zuweisung</i> | <i>Kommando-<br/>Wort</i> |
|-----------------------------------|----------------------------------|---------------------------------|---------------------------|
| <i>Lesen von Kommandos</i>        |                                  |                                 |                           |
| Aliassubstitution                 | –                                | –                               | 1)                        |
| Tilde-Expandierung                | +                                | 2)                              | +                         |
| <i>Expandierung von Kommandos</i> |                                  |                                 |                           |
| Kommandosubstitution              | + 3)                             | +                               | +                         |
| Parametersubstitution             | + 3)                             | +                               | +                         |
| Zerlegen in einzelne Worte        | –                                | –                               | 4)                        |
| Dateinamen-Expandierung           | 5)                               | –                               | + 6)                      |
| Entfernen von Quoting-Zeichen     | +                                | +                               | +                         |

Dabei bedeutet im einzelnen:

- wird nicht durchgeführt
- + wird durchgeführt
- 1) wird für das erste Wort durchgeführt. Wenn der Alias-Text mit einem Leer- oder Tabulatorzeichen endet, dann wird Alias-Substitution auch für das nächste Wort durchgeführt, usw.
- 2) nach = und versionsabhängig nach jedem :
- 3) außer nach << und <<-
- 4) wird nur für Wörter durchgeführt, die aus Kommando- oder Parametersubstitution entstanden
- 5) wird nur durchgeführt, wenn die Dateinamen-Expandierung einen eindeutigen Pfadnamen liefert
- 6) wird nur durchgeführt, wenn die Option **noglob** nicht eingeschaltet ist

## 2.1 Kommandosubstitution

Die **ksh** überprüft jedes Wort, ob es mit **\$(.)** (neue Kommandosubstitution) oder mit **`..`** (alte Kommandosubstitution) geklammert ist. Wenn dies zutrifft, dann führt die **ksh** folgendes aus:

- ▶ Bei der alten Kommandosubstitution werden die Kommandos in **`..`** ausgeführt, wobei die Quoting-Regeln aus Kapitel 5.10 gelten.
- ▶ Bei der neuen Kommandosubstitution wird das Kommando in **\$(.)** ausgeführt; in diesem Fall gelten nicht die Quoting-Regeln aus Kapitel 5.10.

**\$(..)** oder **`..`** wird dann von der **ksh** durch die Ausgabe der darin enthaltenen Kommandos ersetzt, wobei Neuezeile-Zeichen in dieser Ausgabe durch Leerzeichen ersetzt werden. Für diese Ausgabe wird jedoch keine Parameter- oder Kommandosubstitution durchgeführt.

Wenn eine Kommandosubstitution in **".."** angegeben ist, dann wird auch keine Aufteilung in einzelne Wörter und keine Dateinamen-Expandierung durchgeführt.

Es ist darauf hinzuweisen, daß die Kommandos in einer Kommandosubstitution in einer eigenen Subshell-Umgebung (nicht einer wirklichen Subshell) durchgeführt werden und somit nicht die Umgebung der aktuellen Shell verändern, z.B. ändert **\$(cd)** nicht das Working-Directory der aktuellen Shell.

**Beispiel**

```
$ a=10 ↵
$ echo $(echo $a) ↵
10
$ echo $(echo '$a') ↵
$a
$ pwd ↵
/user1/egon/kshueb
$ echo $(cd:pwd) ↵
/user1/egon
$ pwd ↵
/user1/egon/kshueb
$ echo $$ ↵
254
$ echo $(echo $$) ↵
254
$
```

## 2.2 Parametersubstitution

Wenn ein angegebener Parameter (Variable) noch nicht definiert ist, und die Option **nounset** ist gesetzt, wird eine Fehlermeldung auf die Standardfehlerausgabe ausgegeben. Wenn dieser Fehler innerhalb eines Skripts auftritt, so wird das entsprechende Shell-Skript mit einem exit-Status verschieden von 0 beendet.

Führt man die Parametersubstitution innerhalb von **".."** durch, wird nach der Substitution keine Aufteilung in Wörter und keine Dateinamen-Expandierung durchgeführt.

## 2.3 Zerlegen der (aus den ersten beiden Schritten 2.1 und 2.2 entstandenen) Kommandozeile in einzelne Worte

Die **ksh** zerteilt die aus Kommando- und Parametersubstitution entstandene Zeile in einzelne Wörter. Die Trennzeichen für die Wörter sind dabei in der Shell-variablen **IFS** festgelegt.



```
$ IFS=":"
$ var="a:b"
$ set a:b $var "$var" [a:b ist ein Token fuer die ksh]
> for i
> do print "$i"
> done
a:b
a
b
a:b
$
```

## 2.4 Expandierung von Dateinamen

Wenn die Option **noglob** nicht gesetzt ist, wird jedes einzelne Wort nach den Zeichen **\*** **?** und **[** durchsucht. Wenn eines dieser Zeichen in einem Wort gefunden wird, dann wird dieses Wort als *pattern* betrachtet, das eine Vielzahl von Dateinamen abdecken kann. Jedes in der Kommandozeile gefundene pattern wird dann von der Shell expandiert, d.h. durch alle Dateinamen ersetzt<sup>1</sup>, die es abdeckt. Wenn kein Dateiname gefunden werden kann, den ein vorgegebenes pattern abdeckt, wird das entsprechende pattern nicht expandiert und bleibt somit unverändert. Zur Expandierung stehen folgende Metazeichen zur Verfügung:

| Metazeichen | Bedeutung                                         |
|-------------|---------------------------------------------------|
| *           | eine beliebige Zeichenfolge                       |
| ?           | ein beliebiges einzelnes Zeichen                  |
| [..]        | eines der in [..] angegebenen Zeichen             |
| [!..]       | ein Zeichen, welches nicht in [!..] angegeben ist |

In **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, können noch folgende Pattern-Konstruktionen angegeben werden:

| Pattern                   | Bedeutung                                                          |
|---------------------------|--------------------------------------------------------------------|
| ?(pattern[   pattern]...) | deckt kein oder ein Vorkommen der angegebenen Pattern ab.          |
| *(pattern[   pattern]...) | deckt kein, ein oder mehrere Vorkommen der angegebenen Pattern ab. |
| +(pattern[   pattern]...) | deckt ein oder mehrere Vorkommen der angegebenen Pattern ab.       |

1. Alphabetisch sortiert.

| Pattern                 | Bedeutung                                                                        |
|-------------------------|----------------------------------------------------------------------------------|
| @(pattern[ pattern]...) | deckt genau ein Vorkommen der angegebenen Pattern ab.                            |
| !(pattern[ pattern]...) | deckt die Strings ab, die durch keines der angegebenen Pattern abgedeckt werden. |

## 2.5 Entfernen von Quoting

Die Quoting-Zichen \, " und ' werden von der **ksh** entfernt. Quoting-Zichen, die erst durch die Schritte 2.1 und 2.2 entstanden sind, werden dabei jedoch nicht entfernt.

Während eine mit "" bzw. " geklammerte leere Zeichenkette als ein eigenes Wort interpretiert wird, gilt dies nicht für leere Zeichenketten, die ungeklammert sind.

### Beispiel

```
$ x="" y=""
$ print "'Hallo'" ${x}egon${x}
'Hallo' "egon"
$ set "$y" $y
$ print $#
1
$ cat argausg
echo "Die Parameterzahl ist $#"
```

```
echo "Inhalt von \ $1 ist [$1]"
echo "Inhalt von \ $2 ist [$2]"
echo "Inhalt von \ $3 ist [$3]"
echo "Inhalt von \ $4 ist [$4]"
$ chmod u+x argausg
$ LEER=
$ argausg "Leere Argumente" '' "" "$LEER"
Die Parameterzahl ist 4
Inhalt von $1 ist [Leere Argumente]
Inhalt von $2 ist []
Inhalt von $3 ist []
Inhalt von $4 ist []
$ argausg "Leere Argumente" $LEER
Die Parameterzahl ist 1
Inhalt von $1 ist [Leere Argumente]
Inhalt von $2 ist []
Inhalt von $3 ist []
Inhalt von $4 ist []
$
```

## 3. Schritt: Ausführen eines Kommandos

Wichtig ist, daß Alias-Substitution bereits vor diesem Schritt durchgeführt wurde.

Von der **ksh** werden in diesem Schritt Überprüfungen in der folgenden Reihenfolge durchgeführt.

### 3.1 Kommandoname angegeben ?

Falls kein Kommandoname und somit auch keine Kommandoargumente in einer Kommandozeile vorkommen, so handelt es sich bei dieser Zeile um eine

- ▶ Variablen-Zuweisung oder um
- ▶ Ein- / Ausgabe-Umlenkungen

Die **ksh** führt dabei die Variablen-Zuweisung in der aktuellen Shell aus. Ein- / Ausgabe-Umlenkungen dagegen werden in einer eigenen Subshell durchgeführt; deshalb sind in diesem Fall auch nur die beiden Umlenkungs-Operatoren `>` und `>|` sinnvoll, um neue Dateien anzulegen.

Während bei nach dem 3.6.1986 freigegebenen **ksh**-Versionen Variablen-Zuweisungen von links nach rechts abgearbeitet wurden, geschieht dies bei früheren **ksh**-Versionen dagegen von rechts nach links.

Der exit-Status einer solchen Kommandozeile ist 0, wenn sie erfolgreich durchgeführt werden konnte und ansonsten 1. Falls eine Ein- / Ausgabe-Umlenkung innerhalb eines Skripts oder einer Funktion nicht erfolgreich durchgeführt werden kann, wird die Ausführung des Skripts oder der Funktion beendet.

### 3.2 Kommandoname beginnt mit / ?

Wenn der Kommandoname mit `/` beginnt, dann führt die **ksh** das über den absoluten Pfadnamen festgelegte Programm aus.

### 3.3 Reserviertes Wort ?

Wenn es sich bei einem Kommandonamen um ein reserviertes Wort handelt<sup>1</sup>, das eine Konstruktion einleitet, die sich über mehrere Zeilen erstrecken kann (wie z.B. **while** oder **for**), dann liest die **ksh** solange weitere Zeilen ein, bis sie das Kommandoende erkennt.

### 3.4 Builtin-Kommando ?

Die **ksh** führt builtin-Kommandos in der momentanen Shellumgebung aus. Falls jedoch Ein- / Ausgabe-Umlenkungen angegeben sind, so gelten diese außer bei **exec** immer nur für das jeweilige builtin-Kommando und nicht für die aktuelle Shellumgebung. Wird dagegen **exec** mit Ein- / Ausgabe-Umlenkungen, aber ohne sonstige Argumente aufgerufen, werden diese Umlenkungen für die aktuelle Shell ausgewertet.

---

1. Siehe Kapitel 5.16.

Die **ksh** führt für die folgenden builtin-Kommandos eine spezielle Bearbeitung durch:

- ▶ `. : alias break continue eval exec exit export newgrp readonly return shift times trap typeset wait`
- ▶ Variablenzuweisungen werden vor den Ein-/Ausgabe-Umlenkungen ausgewertet, wobei diese Zuweisungen in der aktuellen **ksh** durchgeführt werden, so daß sie auch noch nach Beendigung des builtin-Kommandos gelten.
- ▶ Treten Fehler bei der Ausführung eines der obigen builtin-Kommandos in einem Shell-Skript auf, wird die Skript-Ausführung beendet.

### 3.5 Funktion ?

Wenn ein Kommando kein builtin-Kommando ist, prüft die **ksh** als nächstes, ob es sich um eine Funktion handelt. Ein-/Ausgabe-Umlenkungen, welche bei einem Funktionsaufruf angegeben sind, gelten nur für die Funktion und nicht für die aktuelle Shell. Ein-/Ausgabe-Umlenkungen, die innerhalb einer Funktion mit **exec** vorgenommen werden, haben auch keine Auswirkung auf die aktuelle Shell, sondern nur auf die Funktion.

Die entsprechende Funktion wird in der aktuellen **ksh** ausgeführt; d.h., es wird keine eigene Subshell gestartet. Trotzdem gelten gewisse Konstrukte nur für die Dauer der Funktionsausführung.

Folgendes wird sowohl von der aktuellen **ksh** als auch von einer Funktion gemeinsam benutzt:

- ▶ Werte und Attribute von Variablen, außer bei Verwendung von **typeset** innerhalb einer Funktion
- ▶ Working-Directory
- ▶ Aliase und Funktionsdefinitionen
- ▶ Signalhandler, außer für **EXIT** und **ERR**. Signalhandler werden jedoch nur bei der **ksh**-Version vom 3.6.1986 und früheren Versionen an Funktionen vererbt
- ▶ automatische Parameter

Folgendes wird nicht gemeinsam von der aktuellen Shell und einer Funktion benutzt:

- ▶ Positionsparameter und der automatische Parameter **#** (Anzahl von Positionsparametern)
- ▶ Variablenzuweisungen beim Aufruf einer Funktion
- ▶ Shell-Optionen

- Signal-Handler. Jedoch gilt hier einschränkend, daß Signale, die von der Shell ignoriert werden, auch von einer Funktion ignoriert werden, und daß auf der **ksh**-Version vom 3.6.1986 und früheren Versionen in der aktuellen Shell installierten Signal-Handler (außer **EXIT** und **ERR**) an Funktionen vererbt werden.

### 3.6 Tracked Aliase

Ähnlich der Bourne-Shell verwendet die **ksh** ein »hashing«-Verfahren, um sich an einmal gefundene Kommandos auch später wieder erinnern zu können, ohne wieder von neuem die **PATH**-Directories durchsuchen zu müssen; dazu bedient sie sich sogenannter *Tracked Aliase* (Aliase, die mit **-t** markiert sind).

Die **ksh** generiert für ein einmal aufgerufenes Kommando automatisch ein Tracked Alias. Ein solches Tracked Alias, wird allerdings nur dann generiert, wenn die Option **trackall** eingeschaltet ist, und noch kein gleichnamiges Alias existiert. Einem Tracked Alias wird dann als Wert der Pfadname des entsprechenden Kommandos zugewiesen.

#### Hinweis

Alle momentan definierten Tracked Aliase können mit **alias -t** angezeigt werden. Der Wert eines Tracked Alias wird undefiniert, wenn der Inhalt der Variablen **PATH** geändert wird; das Alias selbst bleibt ein Tracked Alias, dessen Wert erst wieder beim Aufruf des betreffenden Kommandos gesetzt wird.

### 3.7 Programm?

Die **ksh** durchsucht die in **PATH** angegebenen Directories nach den Programmnamen (Kommandonamen) und führt das zugehörige Programm, wenn es existiert und ausführbar ist, in einer eigenen Subshell aus. Ein-/Ausgabe-Umlenkungen und Variablen-Zuweisungen gelten dabei nur für das Programm und nicht für die aktuelle **ksh**.

Wenn die Option **trackall** eingeschaltet ist, so definiert die **ksh** für den Programmnamen automatisch einen Tracked Alias, dem sie als Wert den absoluten Pfadnamen dieses Programms zuweist.

Kann das Programm bei der Suche in den **PATH**-Directories nicht gefunden werden oder fehlen die Ausführrechte für dieses Programm, dann meldet die **ksh** dies auf der Standardfehlerausgabe.

#### Hinweis

Um festzustellen, welches Kommando (bzw. Programm) von der **ksh** ausgeführt wird, kann das builtin-Kommando **whence -v** bzw. das vordefinierte Alias **type** verwendet werden.

```
Beispiel $ set -o trackall
$ pwd
/user1/egon/kshueb
$ x=for
$ type x
x is an alias for for
$ x i in 1 2 3
> do echo $i
> done
1
2
3
$ type cd
cd is a shell builtin
$ alias cd=ls
$ type cd
cd is an alias for ls
$ cd b*
basisname
bname
$ unalias cd
$ type cd
cd is a shell builtin
$ type autoload
autoload is an exported alias for typeset -fu
$ type isnum
isnum is an undefined function
$ type zaehle
zaehle is a function
$ type ls
ls is a tracked alias for /bin/ls
$ alias ls=cat
$ type ls
ls is an alias for cat
$ unalias ls
$ type ls
ls is /bin/ls
$ type add
add is /user1/egon/kshueb/./add
$ type liste
liste not found
$
```

## 5.26 Aufrufsyntax und Umgebung der ksh

In diesem Kapitel wird zunächst die Umgebung (engl. *environment*) einer **ksh** diskutiert, bevor auf die Bearbeitungsschritte eingegangen wird, die von der **ksh** durchgeführt werden, wenn eine **ksh** interaktiv, ein **ksh**-Skript oder eine **ksh**-Funktion aufgerufen werden. Zudem werden hier noch die speziellen Umgebungen für Login-Shells, eingeschränkte Shells, Shell-Skripts, Subshells und Shell-Funktionen vorgestellt.

### 5.26.1 Umgebung

Die Umgebung einer interaktiven **ksh**, eines **ksh**-Skripts oder einer **ksh**-Funktion setzt sich zusammen aus:

- ▶ offenen Dateien
- ▶ Working-Directory
- ▶ Dateikreierungsmaske
- ▶ Ressourcen, welche mit **ulimit** festgelegt werden können.
- ▶ Signal-Handler
- ▶ Shellvariablen mit ihren Werten und ihren Attributen
- ▶ Aliase
- ▶ Funktionen
- ▶ gesetzte Optionen

### 5.26.2 Aufruf einer ksh

Eine **ksh** kann entweder explizit oder implizit über ein Shell-Skript aufgerufen werden. Jedenfalls bewirkt jeder Aufruf einer **ksh**, daß hierfür ein neuer Kindprozeß gestartet wird. Die Umgebung dieses Kindprozesses wird dabei in der folgenden Reihenfolge festgelegt:

- ▶ Vererbung der Umgebung des Elternprozesses
- ▶ Setzen der Positionsparameter
- ▶ Setzen von Optionen
- ▶ Lesen und Ausführen der Environment-Datei
- ▶ Öffnen bzw. Anlegen einer History-Datei

#### Vererbung

Die Umgebung des **ksh**-Kindprozesses wird zunächst generiert, indem gewisse Elemente aus der Umgebung des Elternprozesses kopiert (vererbt) werden.

Kopiert werden dabei:

- ▶ Dateideskriptoren der offenen Dateien
- ▶ Zugriffsrechte für Dateien und Prozesse
- ▶ Working-Directory
- ▶ Dateikreierungsmaske
- ▶ Limits für System-Ressourcen
- ▶ ignorierte Signale
- ▶ Für nicht-ignorierte Signale wird die vorgestellte Signalbehandlung installiert. Für diese Signale kann dann jedoch der Kindprozeß eigene Signal-Handler installieren.
- ▶ Shellvariablen (außer **IFS**) mit ihren Werten und Attributen, für die das **export**-Attribut gesetzt ist. Während bei der **ksh**-Version vom 3.6.1986 und früheren Versionen das **export**-Attribut übernommen und auch die Variable **IFS** vererbt wird, gilt dies nicht bei späteren **ksh**-Versionen.

Nicht kopiert werden dagegen:

- ▶ Alias- und Funktionsdefinitionen. Sogar zum Export markierte Aliase und Funktionen werden nicht vererbt. Um sie einem Kindprozeß verfügbar zu machen, müßten diese in der Environment-Datei angegeben werden.
- ▶ Gesetzte Optionen werden nicht direkt vererbt. Jedoch können Optionen beim Aufruf angegeben werden.

### Positionsparameter

Wenn außer Optionen keine weiteren Argumente angegeben sind, besetzt die **ksh** den Positionsparameter 0 mit "ksh", andernfalls weist sie dem Positionsparameter 0 das erste Argument zu, das keine Option ist.

```
$ ksh
$ echo $0
ksh
$ exit
$ cat ausgab
echo Das erste Argument ist $1
echo Das zweite Argument ist $2
echo Der Skriptname ist $0
$ ksh ausgab one two
Das erste Argument ist one
Das zweite Argument ist two
Der Skriptname ist ausgab
$
```



## Setzen von Optionen

Die mögliche Aufrufsyntax für die **ksh** ist<sup>1</sup>:

**ksh** [**±aefhikmnopr**uvx-] [**±o** *optname*]... [*argument(e)*]

**ksh** [**±aefhikmnopr**uvx-] [**±o** *optname*]... [**-c** *argument*]

**ksh** [**±aefhikmnopr**uvx-] [**±o** *optname*]... [**-s**] [*argument(e)*]

Alle beim builtin-Kommando **set** vorgestellten Optionen können auch beim direkten Aufruf der **ksh** angegeben werden; diese Optionen gelten dann für die Subshell, die durch **ksh** kreiert wird. Die bei **set** erlaubten Optionen werden hier nochmals kurz zusammengefaßt:

| Optionen          | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-a</b>         | ( <b>-o allexport</b> ) automatischer Export für Shell-Variablen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>-e</b>         | ( <b>-o errexit</b> ) Subshell verlassen, wenn ein Kommando nicht erfolgreich ausgeführt werden kann.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>-f</b>         | ( <b>-o noglob</b> ) keine Dateinamen-Expandierung.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>-h</b>         | ( <b>-o trackall</b> ) Für alle aufgerufenen Kommandos wird automatisch ein Tracked Alias definiert.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>-k</b>         | ( <b>-o keyword</b> ) alle Variablen-Zuweisungen auf Kommandozeile gelten für die Umgebung des aufgerufenen Kommandos.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>-m</b>         | ( <b>-o monitor</b> ) Job-Kontrolle einschalten.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>-n</b>         | ( <b>-o noexec</b> ) Kommandos aus einem Shell-Skript nur lesen und auf Syntaxfehler untersuchen, aber nicht ausführen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>-o optname</b> | Für <i>optname</i> kann neben den bei den Optionen bereits erwähnten Optionsnamen noch folgendes angegeben werden: <ul style="list-style-type: none"> <li><b>bgnice</b> alle Hintergrund-Jobs mit einer niedrigeren Priorität ausführen.</li> <li><b>emacs</b> schaltet builtin-Editor <b>emacs</b> ein</li> <li><b>gmacs</b> schaltet builtin-Editor <b>gmacs</b> ein</li> <li><b>ignoreeof</b> Eine interaktive <b>ksh</b> kann nur mit <b>exit</b> (nicht mit <b>Strg-d</b>) verlassen werden.</li> <li><b>markdirs</b> An alle Directory-Namen, welche aus Dateinamen-Expandierung resultieren, einen Slash / anhängen</li> <li><b>noclobber</b> Bei &gt; werden existierende Dateien nicht überschrieben.</li> <li><b>nolog</b> Funktionsdefinitionen nicht in der History-Datei speichern.</li> </ul> |

1. [**±optname**] kann bei einem **ksh**-Aufruf öfters angegeben werden.

| Optionen  | Bedeutung                                                                                                                                                                                                              |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           | <b>vi</b> schaltet den builtin-Editor <b>vi</b> ein                                                                                                                                                                    |
|           | <b>viraw</b> Jedes eingegebene Zeichen wird so interpretiert, als ob es im <b>vi</b> -Modus eingegeben wurde.                                                                                                          |
| <b>-p</b> | Diese Option hat unterschiedliche Bedeutungen auf den einzelnen <b>ksh</b> -Versionen:<br><br>( <b>-o privileged</b> ) <b>ksh</b> -Versionen nach 3.6.1986<br>( <b>-o protected</b> ) <b>ksh</b> -Version vom 3.6.1986 |
| <b>-t</b> | Nur ein Kommando ausführen.                                                                                                                                                                                            |
| <b>-u</b> | ( <b>-o nounset</b> ) Zugriff auf nicht gesetzte Shell-Variablen als Fehler wer-<br>ten.                                                                                                                               |
| <b>-v</b> | ( <b>-o verbose</b> ) alle Shell-Eingabezeilen – so wie sie gelesen werden – aus-<br>geben.                                                                                                                            |
| <b>-x</b> | ( <b>-o xtrace</b> ) jedes einfache Kommando unmittelbar vor seiner Ausfüh-<br>rung (nach Parametersubstitution, Kommandosubstitution und Datei-<br>namen-Expandierung) auf der Standardfehlerausgabe anzeigen.        |
| <b>-</b>  | (Minuszeichen) schaltet zunächst die beiden Optionen <b>-x</b> und <b>-v</b> aus<br>und behandelt den Rest der Kommandozeile als normale Argumente,<br>selbst wenn diese mit <b>-</b> (Minus) beginnen.                |
| <b>--</b> | (doppeltes Minuszeichen) hat keine Auswirkung auf die Optionen.<br>Wenn nach <b>--</b> keine weiteren Argumente angegeben sind, so werden<br>die Positionsparameter gelöscht.                                          |

Allgemein gilt, daß die Optionenangabe immer dann als abgeschlossen gilt, wenn ein Argument, das nicht mit **+** oder **-** beginnt, oder ein einfaches bzw. dop-  
pelt Minuszeichen vorkommt.

Folgen also nach einem **-** oder **--** weitere Argumente, so werden diese immer als  
normale Argumente gewertet, selbst wenn sie mit einem **-** (Minus) beginnen.  
Um den Positionsparameter **\$1** mit einem String zu besetzen, der mit **-** (Minus)  
beginnt, muß **-** bzw. **--** verwendet werden, z.B.

```
$ set -- -o xtrace
$ echo $1
-o
$ echo $2
xtrace
$
```

Optionen, die ein Benutzer immer automatisch einschalten lassen möchte, soll-  
ten üblicherweise in *\$HOME/.profile* oder in der Environment-Datei gesetzt  
werden.

Der automatische Parameter **\$-** enthält die gesetzten Optionen für die momentan aktive Shell:

```
$ echo $-
ismh
$
```

Um sich den Zustand aller Optionen anzuzeigen, müßte

**set -o**

aufgerufen werden. In diesem Fall werden alle Optionsnamen mit ihrem Zustand (*on* oder *off*) angezeigt.

Zusätzlich können bei einem **ksh**-Aufruf noch die folgenden Optionen angegeben werden:

| Optionen                            | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-c</b> <i>argument</i>           | Die als <i>argument</i> angegebene Kommandoliste wird gelesen und ausgeführt; z.B:<br><br>ksh -c 'ls   wc -l'<br><br>gibt die Anzahl von Dateien im Working-Directory aus.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>-s</b><br>[ <i>argument(e)</i> ] | bewirkt, daß eine interaktive Subshell gestartet wird, die Kommandos von der Standardeingabe liest. Wenn <i>argument(e)</i> angegeben sind, dann werden diese der Subshell als Positionsparameter übergeben. Ausgaben der Subshell werden in die mit Dateideskriptor 2 spezifizierte Datei (Standardfehlerausgabe) geschrieben. Diese Option wird automatisch eingeschaltet, wenn die Option <b>-c</b> nicht angegeben ist. Falls die Option <b>-c</b> angegeben ist, wird die Option <b>-s</b> ignoriert.<br><br><b>Beispiel:</b><br><br>\$ pwd<br>/user1/egon/kshueb<br>\$ ksh -s egon \$(pwd)<br>\$ echo "Hallo, \$1"<br>Hallo, egon<br>\$ echo \$2<br>/user1/egon/kshueb<br>\$ exit<br>\$ |
| <b>-i</b>                           | macht die Shell interaktiv, d.h., daß die Standardeingabe und die beiden Ausgabekanäle auf die Dialogstation eingestellt sind. Zudem gilt: <ul style="list-style-type: none"><li>– das Signal <b>TERM</b> wird in dieser Shell ignoriert.</li><li>– das Signal <b>INT</b> wird abgefangen und bewirkt den Abbruch des</li></ul>                                                                                                                                                                                                                                                                                                                                                               |
| <b>-r</b>                           | die entsprechende Shell wird als »restricted Shell« (eingeschränkte Shell) aufgerufen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

Die **ksh** schaltet bis auf einige Ausnahmen alle nicht explizit eingeschalteten Optionen aus. Die Ausnahmen sind dabei:

- ▶ Die **ksh** schaltet automatisch die Option **interactive** ein, wenn außer Optionen keine anderen Argumente angegeben sind; in diesem Fall werden die Standardeingabe und die Standardausgabe auf die Dialogstation eingestellt.
- ▶ Die **ksh** schaltet die Option **trackall** ein, wenn sie nicht interaktiv ist.
- ▶ Die **ksh** schaltet die Option **monitor** ein, wenn die Option **interactive** eingeschaltet ist und das jeweilige System über eine Job-Kontrolle verfügt.
- ▶ Die **ksh** schaltet abhängig vom Inhalt der beiden Variablen **EDITOR** und **VISUAL** einen der builtin-Editoren **emacs**, **gmacs** oder **vi** ein.
- ▶ **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, schalten automatisch die Option **bgnice** ein.
- ▶ Die **ksh** schaltet die Option **restricted** ein, wenn die Variable **SHELL** oder der Dateiname der aufgerufenen **ksh** ein "r" enthält.
- ▶ Die **ksh** schaltet die Option **protected** ein, wenn die effektive UID des Prozesses nicht gleich der realen UID bzw. die effektive GID nicht gleich der realen GID des Prozesses ist. Die Option **protected** ist jedoch nur auf der **ksh**-Version vom 3.6.1986 verfügbar.

### Lesen und Ausführen der Environment-Datei

Wenn die Option **privileged**

- ▶ ausgeschaltet ist, dann liest die **ksh** beim Aufruf die Environment-Datei, welche über die Variable **ENV** festgelegt ist, und führt die darin enthaltenen Kommandos aus.
- ▶ eingeschaltet ist, dann liest die **ksh** die Datei */etc/suid\_profile*, wenn diese existiert, und führt die darin enthaltenen Kommandos aus.

Um Funktionsdefinitionen und/oder Alias-Definitionen auch für andere **ksh**-Aufrufe zur Verfügung zu stellen, muß wie folgt vorgegangen werden:

- ▶ Alias- und Funktions-Definitionen in die Environment-Datei schreiben.
- ▶ Die entsprechenden Aliase und Funktionen zum Export markieren.
- ▶ Der Variable **ENV** den Pfadnamen der Environment-Datei zuweisen.
- ▶ **ENV** für den Export markieren.

Alle Alias- und Funktionsdefinitionen, für die nicht das **export**-Attribut gesetzt ist, werden bei nicht-interaktiven Shells entfernt.

### Anlegen bzw. Öffnen der History-Datei

Für interaktive Shells eröffnet bzw. kreiert die **ksh** die History-Datei<sup>1</sup> entweder beim ersten Lesen einer Funktionsdefinition (wenn die Option **nolog** nicht eingeschaltet ist) oder nachdem die Environment-Datei vollständig gelesen und ausgeführt ist, je nachdem, was zuerst zutrifft.

### 5.26.3 Login-Shells

Die Login-Prozedur ruft die Korn-Shell mit

#### **exec -ksh**

auf. Das Minuszeichen – bewirkt, daß in diesem Fall eine Login-Shell gestartet wird. Eine Login-Shell unterscheidet sich von einer normalen Shell nur in den folgenden Punkten:

- ▶ Es können beim Aufruf außer Optionen keine Argumente angegeben werden.
- ▶ Es werden zuerst die einzelnen Kommandos aus der Datei */etc/profile* gelesen und ausgeführt, wenn diese Datei existiert.
- ▶ Wenn die Option **privileged** nicht eingeschaltet ist, dann werden die einzelnen Kommandos aus der Datei **\$HOME/.profile** gelesen und ausgeführt, wenn diese Datei existiert.

### 5.26.4 Restricted Shells

Die Umgebung einer eingeschränkten **ksh** (engl. *restricted shell*) ist sehr stark gegenüber der Umgebung einer normalen Shell limitiert. So kann folgendes in einer restricted **ksh** nicht durchgeführt werden:

- ▶ Wechseln des Working-Directorys
- ▶ Verändern der Werte der Shellvariablen **SHELL**, **ENV** und **PATH**
- ▶ Aufruf eines Kommandos über einen Pfadnamen, in dem / vorkommt.
- ▶ Umlenken der Ausgabe mit **>**, **>|**, **<>** oder **>>**.

Während des Lesens und der Ausführung der Datei *.profile* und der Environment-Datei gelten diese Einschränkungen für eine restricted **ksh** nicht.

Wenn eine restricted **ksh** ein Shell-Skript ausführt, dann findet die Ausführung dieses Shell-Skripts in einer normalen, nicht eingeschränkten Shell-Umgebung statt.

---

1. Der Name dieser Datei kann über die Variable **HISTFILE** festgelegt werden. Falls diese Variable **HISTFILE** nicht gesetzt ist, verwendet die **ksh** die Datei **\$HOME/.sh\_history** als History-Datei.

### 5.26.5 Shell-Skripts

Ein **ksh**-Skript kann auf zwei verschiedene Arten aufgerufen werden:

- ksh** skriptname ... (1)
- skriptname (2)

Bei der Aufrufform (2) findet anders als in der Bourne-Shell kein neuer Aufruf von **ksh** statt. Ausnahmen dazu werden im nächsten Kapitel bei den Subshells gezeigt. Zudem ist die zweite Aufrufform auch schneller als die erste. Außerdem werden bei der Aufrufform (2) die für den Export markierten Funktionen, Aliase und Arrays an das Skript vererbt.

Ein Shell-Skript wird üblicherweise immer dann durch einen neuen Aufruf von **ksh** ausgeführt, wenn:

- ▶ es Leserecht, aber kein Ausführrecht besitzt.
- ▶ das *setuid*- oder *setgid*-Bit gesetzt ist.
- ▶ die erste Zeile mit **#!** beginnt, und danach der Pfadname für eine bestimmte **ksh** angegeben ist; dieser Pfadname legt dann das **ksh**-Programm fest, welches dieses Skript ausführen soll. Diese Möglichkeit ist jedoch nicht auf allen Systemen verfügbar.

Wenn ein Skript nicht durch einen neuen Aufruf von **ksh** zur Ausführung gebracht wird, so gilt:

- ▶ Die Environment-Datei wird nicht gelesen und nicht ausgeführt.
- ▶ Die History-Datei wird nicht vererbt.
- ▶ Zum Export markierte Aliase, Funktionen und Arrays werden vererbt. Alle anderen Aliase, Funktionen und Arrays werden nicht vererbt.
- ▶ Variablen-Attribute werden vererbt. Auf **ksh**-Versionen, die nach dem 3.6.1986 freigegeben wurden, werden Variablen-Attribute auch bei einem eigenen **ksh**-Aufruf vererbt.

**Beispiel**

```
$ alias -x dir=ls
$ cat lst
dir b*
$ ksh lst
lst: dir: not found [exportiertes Alias dir wurde nicht vererbt]
$ chmod u+x lst
$ lst
basisname
bname
$
```

Ein Shell-Skript wird solange ausgeführt, bis

- ▶ alle seine Kommandos ausgeführt sind, oder
- ▶ **exit** ausgeführt wird, oder
- ▶ **return** außerhalb einer Funktion ausgeführt wird, oder
- ▶ **exec** mit einem Argument ausgeführt wird; in diesem Fall wird das Shell-Skript durch das bei **exec** angegebene Programm überlagert, oder
- ▶ ein Fehler bei bestimmten builtin-Kommandos<sup>1</sup> auftritt, oder
- ▶ ein Signal eintrifft, das nicht ignoriert wird oder für das kein eigener Signal-Handler installiert wurde, oder
- ▶ ein Kommando einen exit-Status verschieden von 0 liefert, wenn die Option **errexit** eingeschaltet ist und kein Signal-Handler für **ERR** installiert ist.

### 5.26.6 Subshells

Eine Subshell ist ein Kindprozeß zur Elternshell. Bei der Kreierung einer Subshell wird die Umgebung des Elternprozesses kopiert. Änderungen, welche an der Umgebung einer Subshell vorgenommen werden, haben somit keinen Einfluß auf die Umgebung der Elternshell.

Die **ksh** kreiert eine Subshell für

- ▶ Kommandoklammerung: **(kdo liste)**
- ▶ Ko-Prozesse
- ▶ Hintergrund-Prozesse
- ▶ jedes Element einer Pipeline, außer für das letzte. Auf **ksh**-Versionen vom 3.6.1986 und früheren Versionen wurde auch für das letzte Element einer Pipeline eine Subshell gestartet.
- ▶ Bei einem neuen Aufruf einer **ksh**

---

1. Siehe Schritt 3.4 in Kapitel 5.25.

## 5.26.7 Shell-Funktionen

**ksh**-Funktionen teilen sich ihre Umgebung bis auf die folgenden Ausnahmen mit dem aufrufenden Prozeß:

- ▶ Variablen-Zuweisungen, die beim Funktionsaufruf angegeben sind.
- ▶ gesetzte Optionen.
- ▶ Variablen, die in der Funktion mit **typeset** deklariert werden.
- ▶ Signal-Handler. Auf der **ksh**-Version vom 3.6.1986 und früheren Versionen teilen sich die Funktionen die einzelnen Signal-Handler (außer für **EXIT**) mit dem Aufrufer.

## 5.27 Einrichten einer persönlichen Arbeitsumgebung

Hier wird anhand von Beispielen gezeigt, wie ein Benutzer seine persönliche Arbeitsumgebung den eigenen Anforderungen anpassen kann. Dazu werden Beispiele zu einem möglichen Aussehen der Datei *.profile* und der Environment-Datei gegeben. Danach werden noch einmal alle bisher erarbeiteten und auch noch einige neue **autoload**-Funktionen vorgestellt, bevor dann Hinweise gegeben werden, wie die Ablaufgeschwindigkeit von Skripts oder Funktionen verbessert werden kann.

### 5.27.1 Die Datei *.profile*

Wenn ein Benutzer sich am System anmeldet, so startet die login-Prozedur die **ksh**<sup>1</sup> mit *-ksh*; dieses führende Minuszeichen zeigt an, daß es sich um eine Login-Shell handelt. Deshalb führt diese Shell auch zuerst die Datei */etc/profile*, wenn existent, und dann die Datei **\$HOME/.profile** aus.

Die Datei *.profile* (im Home-Directory) wird üblicherweise verwendet, um

- ▶ Variablen zu setzen und zu exportieren, welche von vielen Programmen verwendet werden. Ein typisches Beispiel für eine solche Variable ist **TERM**.
- ▶ nützliche Optionen (wie **trackall**, **ignoreeof**, usw.) für die Login-Shell immer automatisch einschalten zu lassen.
- ▶ Aktionen festzulegen, die immer am Anfang oder Ende einer UNIX-Sitzung durchzuführen sind, wie z.B. Einbau eines automatischen Erinnerungsservice oder beim Beenden der UNIX-Sitzung das Working-Directory für das nächste Anmelden merken (**trap '...' EXIT**).

---

1. Wenn diese vom Systemadministrator als Login-Shell eingetragen ist; die Login-Shell jedes einzelnen Benutzers ist in der Datei */etc/passwd* angegeben.



Nachfolgend wird nun ein Beispiel für das Aussehen einer *.profile*-Datei für die **ksh** gegeben. Dabei handelt es sich lediglich um einen Vorschlag. Jeder Benutzer kann diese Datei dann seinen eigenen Bedürfnissen anpassen:

```
#-----
Beispiel fuer ein login .profile in der KornShell
#-----

#.... Setzen und Exportieren von Variablen
#.....
export PATH=/bin:/usr/bin:.
export ENV=$HOME/.ksh_env
export SHELL=/bin/ksh
export FPATH=$HOME/flib
export HISTSIZE=150
export MAIL=/usr/spool/mail/$LOGNAME
export MAILPATH=~uucp/mail:$MAIL
export MAILCHECK=100
export EDITOR=$(whence vi)
export TERM=ansi
export CDPATH=":$HOME:/usr:/user1/graphgruppe"
export PS3='Deine Wahl: '
export PS4='Zeile $LINENO: '
export PS1='!>$PWD: '

Andere beliebte Angaben fuer PS1 sind:
work. dir. immer nur relativ zur home dir.
ausgeben ---> kein ueberlanger Prompt
PS1='!>${PWD#/$HOME/}: '
#
Aktuelle Tageszeit anzeigen
export SECONDS="$(date '+3600*%H+60*%M+%S')"
```

```
typeset -Z2 _std _min
_st="(SECONDS/3600)%24"
_mi="(SECONDS/60)%60"
_z='${_x[(_min=_mi)==(_std=_st)]}$ _std:$_min'
PS1="($_z)"$ _std:$_min: '

#.... Setzen von nuetzlichen Optionen
#.....
set -o trackall -o ignoreeof -o nolog

#.... Terminal mit eigenen Tasten belegen
#.....
stty susp '^z'

#.... Automatischer Erinnerungs-Service
#.....
calendar

#.... Ausfuehren der Kommandos aus .logout
```

```
#.....
. $HOME/.logout

#.... Beim Abmelden Kommandos fuer das naechste
#.... Anmelden in .logout hinterlegen
#.....
function abmeld
{
 echo echo Letzte Abmeldung: $(date)
 echo 'echo "\n--- Heute ist $(date) ----"'
 echo echo ---- Working-Directory: $PWD ----
 echo cd $PWD
}
trap 'abmeld >$HOME/.logout; exit' EXIT
```

### 5.27.2 Die Environment-Datei

Immer wenn die **ksh** aufgerufen wird, führt sie die Kommandos der Environment-Datei aus. Der Name der Environment-Datei ist in der Variablen **ENV** gespeichert. Die Environment-Datei wird üblicherweise benutzt, um

- ▶ Aliase und Funktionen (einschließlich **autoload**-Funktionen) automatisch für die Login-Shell und für Subshells definieren zu lassen.
- ▶ Bestimmte Optionen für alle **ksh**-Aufrufe automatisch setzen zu lassen.
- ▶ Variablen automatisch für alle möglichen **ksh**-Aufrufe setzen zu lassen.

Da jeder Aufruf der **ksh** das Lesen und Ausführen der Environment-Datei impliziert, ist es empfehlenswert, diese Datei nicht zu überladen, um somit eine bessere Ablaufgeschwindigkeit für **ksh**-Aufrufe zu erzielen. So ist z.B. nicht ratsam, alle im Laufe der Zeit entwickelten nützlichen Funktionen in der Environment-Datei zu definieren, sondern die weniger häufig gebrauchten Funktionen mit dem **autoload**-Mechanismus bei Notwendigkeit nachdefinieren zu lassen.

Ein mögliches Aussehen für eine Environment-Datei ist z.B.:

```
$ cat $ENV 
#-----
Beispiel fuer eine Environment-Datei
#-----

#..... Alias-Definitionen
#.....
alias -x repeat='function _repeat {'
alias -x from='}; _from'
alias -x pot='set -o noglob; _pot'
alias -x sh=${SHELL:-/bin/sh}
alias -x suspend='kill -STOP $$'
alias -x stop='kill -STOP '
```

```
alias -x cx='chmod u+x'
alias -x h='fc -l'
if [[-f $(whence more)]] # more wird oft anstelle
then : # von pg verwendet.
else alias -x more=$(whence pg)
fi

#..... autoload-Funktionen
#.....
autoload isprim # Primzahl ?
autoload isnum # Integer-Zahl ?
autoload _pot # Potenz berechnen
autoload zaehle # zaehle start ende schrittweite
 # realisiert einfache for-Schleife
autoload _from # fuer Realisierung einer
 # repeat-Schleife
autoload substr # substr string pos [zahl]
 # schneidet aus string ab Position
 # 'pos' einen Teilstring mit 'zahl'
 # Zeichen bzw. Reststring heraus
#..... Setzen von Optionen
#.....
set -o trackall -o ignoreeof -o nolog

#..... Funktions-Definitionen
#.....
typeset -xf dirs pushd popd ll

function dirs
{
 print - "${PWD};$DS"
}

function pushd
{
 DS="$PWD;$DS"
 cd $1
 dirs
}

function popd
{
 if ["$DS"]
 then
 cd ${DS%%\;*}
 DS=${DS#*\;}
 dirs
 else
 echo "Stack ist leer"
 fi
}
```

```
function ll
{
 typeset i # Lokale Variable
 set -o markdirs
 if [[$# = 1 && -d $1]]
 then cd "$1"
 trap 'cd $OLDPWD' EXIT
 set --
 fi
 PS3= # es wird keine interaktive Eingabe
 # erwartet.
 select i in ${@-*} # Ausgabe aller Dateien mit
 do : # Numerierung
 done </dev/null 2>&1
}
$. $ENV
$ pwd
/user1/egon/kshueb

$ ll
1) a* 15) diriname 29) pruef
2) a.out 16) eingabe 30) ps.txt
3) abc 17) einles 31) rechne
4) abc2 18) fiba 32) selecttest
5) add 19) hanoi 33) seltext
6) addiere.c 20) hexz 34) stdio.h
7) argausg 21) homedir 35) sum_ungerad
8) ausgab 22) kexpr 36) teiler
9) basisname 23) logfile 37) teiler.txt
10) bname 24) lotto 38) teiler1.txt
11) ccvi 25) lst 39) vonhinten
12) core 26) mein_cd 40) wartmsg
13) cph 27) neudat 41) zaehle.txt
14) datanal 28) num 42) zufall
$
```

### 5.27.3 Autoload-Funktionen

Eine **autoload**-Funktion wird erst dann definiert, wenn sie das erstmal aufgerufen wird. Dies hat den Vorteil, daß die **ksh** keine Zeit mit dem Lesen von Funktionsdefinitionen vergeudet, die niemals aufgerufen werden. Beim ersten Aufruf durchsucht die **ksh** die in der Shellvariablen **FPATH** enthaltenen Directories nach einem Dateinamen, welcher der Name der aufgerufenen Funktion ist. Findet die **ksh** die gleichnamige Datei, so liest sie zunächst den Inhalt dieser Datei und führt dann die entsprechende Funktion aus.

Nachfolgend werden nun einige nützliche **autoload**-Funktionen gegeben, die in den gleichnamigen Dateien im »Funktionen-Directory« (hier: `/user1/egon/flib`) untergebracht sind.

Bis auf die Funktion **substr** wurden alle hier vorgestellten Funktionen bereits in den vorhergehenden Kapiteln entworfen. Sie werden hier nochmals wiederholt:

```
$ echo $FPATH↵
/user1/egon/flib
$ cd↵
$ cd flib↵
$ pwd↵
/user1/egon/flib
$ cat _from↵
#----- _from -----
realisiert Schleifen wie
for i='$1' to '$3' step '$5'
#
function _from
{ typeset var="${1%=*}"
 integer incr="${5:-1}" "$1"
 while (($var<=$3))
 do _repeat
 let $var=$var+incr
 done
}
$ cat zaehle↵
#----- zaehle -----
realisiert eine einfache for-Schleife:
zaehle $1 $2 $3 entspricht
for (x=$1 ; x<=$2 ; x=x+$3)
#
function zaehle
{
 integer start=${1:-1} ende=${2:-10} step=${3:-1}
 integer i=start
 while ((i<=ende))
 do print - $i
 i=i+step
 done
}
$ cat isnum↵
-- isnum -----
Rueckgabewert: 0 (TRUE), wenn erstes Argument
nur aus Ziffern besteht.
1 (FALSE), ansonsten
function isnum
{ typeset para="$1" zahl=1 # zahl ist boole'sche Var.
 while true
 do case $para in
 [0-9]*) para=${para#?}
 zahl=0;;
 "") return $zahl;;
 *) return 1;;
 esac
}
```

```
done
}
$ cat isprim ↵
-- isprim -----
#
Rueckgabewert: 0 (TRUE), wenn erstes Argument
eine Primzahl ist
1 (FALSE), ansonsten
function isprim
{
 set -- $(factor $1 2>/dev/null)
 ((rueckgabe=$((2)) # wenn Primzahl, dann liefert
 # factor 2 Zahlen
 return $rueckgabe
}
$ cat _pot ↵
#---- _pot -----
realisiert den Potenzoperator
#
function _pot
{
 trap 'set +o noglob' EXIT
 if [[$# -ne 2]]
 then echo "usage: $0 operand potenz" >&2
 return 1
 fi

 integer op=$1 pot=$2 ergebn=1 i=1
 if isnum $op && isnum $pot
 then
 while ((i<=pot))
 do ergebn=ergebn*op
 i=i+1
 done
 print $ergebn
 else
 echo "Nur Integer-Zahlen erlaubt" >&2
 return 1
 fi
}
$ cat breakpoint ↵
#---- breakpoint -----
ermoglicht interaktive Eingaben beim
Debuggen eines Shell-Skripts.
#
function breakpoint
{
 typeset zeile
 while true
 do echo -n "Zeile $lineno :ksh-dbg>> "
 read -r zeile
 if [[-z "$zeile"]]

```

```
 then return
 fi
 eval "$zeile"
 done
}
$
```

Die neue Funktion **substr** liefert einen Substring aus einem vorgegebenen String, der als erstes Argument anzugeben ist. Das zweite Argument legt die Position fest, ab welcher der Substring »herauszuschneiden« ist. Ist auch noch ein drittes Argument angegeben, so legt dies die Zahl der »herauszuschneidenden« Zeichen fest. Ist kein drittes Argument angegeben, so wird ab der angegebenen Position der Rest des Strings als Substring geliefert:

```
$ cat substr ↵
#---- substr -----
#
Aufrufsyntax: substr $1 $2 [$3]
substr liefert einen Substring aus $1.
$2 legt dabei die Position fest, ab der dieser
Substring "herauszuschneiden" ist.
Ist $3 angegeben, so legt dieses die Anzahl der
herauszuschneidenden Zeichen fest; fehlt $3, so
werden ab $2 alle Zeichen herausgeschnitten.
#
function substr
{
 integer strlaenge

 if (($# != 2 && $# != 3))
 then print "usage: $0 string pos [anzahl]" >&2
 exit 1
 fi
 string="$1"
 strlaenge=${#string}
 if isnum $2
 then integer r_anz
 ((r_anz=strlaenge - $2 + 1))
 typeset -RZ$r_anz substring="$string"
 else print - "$2 ist keine Integer-Zahl" >&2
 exit 1
 fi

 if (($# == 3))
 then if isnum $3
 then typeset -L$3 substring="$substring"
 else print - "$3 ist keine Integer-Zahl" >&2
 exit 1
 fi
 fi
fi
```

```
 echo "$substring"
}
$ substr Manfred 5 ↵
red
$ substr "Helmut Herold" 4 3 ↵
mut
$
```

### 5.27.4 Verbesserte Ablaufgeschwindigkeit für Skripts und Funktionen

Um die Ablaufgeschwindigkeit von **ksh**-Skripts oder -Funktionen zu verbessern, sollten die folgenden Ratschläge berücksichtigt werden:

- ▶ Wenn möglich, sollten builtin-Kommandos verwendet werden. Builtin-Kommandos laufen wesentlich schneller ab als andere Programme.
- ▶ Funktionen sind langsamer als builtin-Kommandos.
- ▶ Parametersubstitution, Arithmetische Berechnungen und Dateinamen-Expansion sollten, wenn möglich, immer anstelle von Kommandosubstitution verwendet werden:

```
$ time echo $(ls) ↵
....
....

real 0m0.35s
user 0m0.20s
sys 0m0.13s
$ time echo * ↵
....
....

real 0m0.01s
user 0m0.01s
sys 0m0.01s
$
```

- ▶ Feste Schleifen-Bedingungen sollten vor der Schleife ausgewertet werden, damit nicht bei jedem Schleifendurchlauf eine erneute Auswertung notwendig wird:

```
$ time for i in $(ls) ↵
> do echo $i ↵
> done ↵
.....
.....

real 0m0.48s
user 0m0.26s
```



```
sys 0m0.15s
$ set - *
$ time for i in *
> do echo $i
> done
.....
.....

real 0m0.05s
user 0m0.06s
sys 0m0.00s
$
```

- Es sollte immer **\$(< datei)** anstelle von **\$(cat datei)** oder **`cat datei`** verwendet werden.
- **{..}** sollte **(..)** vorgezogen werden, da die **ksh** für **(..)** immer eine Subshell startet. Für **{..}** startet die **ksh** nur eine Subshell, wenn dies erforderlich ist.
- Die Umlenkung für eine ganze **for**-, **while**- oder **until**-Schleife ist schneller als die Angabe dieser Umlenkung bei einzelnen Kommandos in der Schleife:

```
$ time for i in *
> do echo $i >tmp
> done

real 0m0.48s
user 0m0.10s
sys 0m0.40s
$ time for i in *
> do echo $i
> done >tmp

real 0m0.10s
user 0m0.08s
sys 0m0.03s
$
```

- Der Zugriff auf eine Integer-Variable in einem arithmetischen Ausdruck ist schneller als die Verwendung von Parametersubstitution (Vorstellen von **\$**), um auf den Zahlenwert einer Variable ohne das Integer-Attribut zuzugreifen.

## 5.28 Anwendungsbeispiele

Hier werden zum Teil etwas umfangreichere **ksh**-Skripts vorgestellt, um die vielen Einsatzmöglichkeiten der **ksh**-Konstrukte aufzuzeigen, was auch zur Vertiefung und Wiederholung der wichtigsten **ksh**-Elemente beitragen soll. Außerdem dienen die hier vorgestellten Skripts auch zum Lernen des Umgangs mit bereits vorhandenen Skripts. Eine Fähigkeit, die bei der tagtäglichen Softwarepraxis nicht zu unterschätzen ist, da es immer wieder vorkommen wird, daß bereits bestehende Skripts verbessert oder geändert werden müssen.

### 5.28.1 Das Spiel moo

Das Spiel **moo** ähnelt dem bekannten Spiel »Mastermind«. **moo** denkt sich eine vierstellige Zahl, die es zu erraten gilt. Als Regel gilt, daß eine Ziffer nie mehr als einmal in dieser gedachten Zahl vorkommt. Die Ziffer 0 kann wie jede andere Ziffer an jeder beliebigen Stelle vorkommen. Beispiele für erlaubte Zahlen wären also 2791, 0823 usw. Nicht erlaubt wären dagegen 2723, 3889 usw.

Nach jedem Rateversuch gibt **moo** folgendes aus:

*m* Bullen; *n* Kuehe

*m* gibt dabei die Anzahl von richtigen Ziffern an richtigen Positionen und

*n* gibt die Anzahl von richtigen Ziffern an falschen Positionen an.

```
$ cat moo ↵
#-----
Spiel moo: aehneln dem bekannten Mastermind-Spiel.
moo generiert eine 4-stellige Zufallszahl,
welche der Spieler dann ermitteln muss.
Nach jedem Versuch gibt moo folgendes aus:
m Bullen; n Kuehe
m = Anzahl von richtigen Ziffern
an richtigen Positionen
n = Anzahl von richtigen Ziffern
an falschen Positionen
#-----

#--- Signal-Handler einrichten ---
#-----
trap 'echo -n "gesuchte Zahl war "
 for i in 1 2 3 4
 do echo -n ${ziffer[i]}
 done; echo; exit' HUP INT QUIT TERM

#--- Variablen-Deklarationen ---
#-----
integer bullen kuehe # Zaehl-Variablen
integer versuche=0 # zaehlt die Versuche
```

```
integer ziffer[4] # 4-stellige Zufallszahl
integer bziffer[4] # vorgeschlagene Benutzer-Zahl
typeset -l1 zeich # fuers Zerlegen der eingegeb.
 # Zahl in einzelne Ziffern
integer i j # Laufvariablen

#---- Funktions-Deklarationen -----
#-----

#.....
generiert 4-stellige Zufallszahl ins Array ziffer,
wobei jede Ziffer eindeutig ist.
erlaubte Zahlen sind z.B. 0398, 1526, 8463, usw.
nicht erlaubte Zahlen sind z.B. 1231, 1022, 8988,...
#.....
function bestimme_zufalls_zahl
{
 ((ziffer[1]=RANDOM % 10))
 for i in 2 3 4
 do ((ziffer[i]=RANDOM % 10))
 j=1
 while ((j<i))
 do if ((ziffer[i] == ziffer[j]))
 then ((ziffer[i]=RANDOM % 10))
 j=1
 else j=j+1
 fi
 done
 done
}

#.....
Eingegebene Benutzerzahl bezueglich
Treffern (bullen und kuehe) analysieren
#.....
function zahl_analyse
{
 integer i j
 bullen=0
 for i in 1 2 3 4 # bullen zaehlen
 do if ((ziffer[i] == bziffer[i]))
 then bullen=bullen+1
 fi
 done
 kuehe=0
 for i in 1 2 3 4
 do for j in 1 2 3 4
 do if ((ziffer[i]==bziffer[j]))
 then kuehe=kuehe+1
 fi
 done
 done
}
```

```
kuehe=kuehe-bullen
echo "$bullen Bullen; $kuehe Kuehe\n"
}

#---- main -----
#-----

#... Zufallszahl bestimmen
bestimme_zufalls_zahl

until ((bullen==4))
do
 #... Zahl einlesen und pruefen
 read zahl?"Dein Vorschlag: "
 if [[$zahl = [0-9][0-9][0-9][0-9]]]
 then versuche=versuche+1
 else echo "ungueltige Zahl" >&2
 continue
 fi
 #... Zahl in Array bziffer speichern
 #... und analysieren
 for i in 1 2 3 4
 do zeich=$zahl
 bziffer[i]=$zeich
 zahl=${zahl#?}
 done
 zahl_analyse
done
echo "\nDu hast die Zahl nach $versuche Versuchen gefunden"
#-----
$ cx moo [↵] [cx = Alias für chmod u+x]
$ moo [↵]
Dein Vorschlag: 1234 [↵]
0 Bullen; 1 Kuehe

Dein Vorschlag: 2345 [↵]
0 Bullen; 2 Kuehe

Dein Vorschlag: 3467 [↵]
0 Bullen; 2 Kuehe

Dein Vorschlag: 5673 [↵]
2 Bullen; 1 Kuehe

Dein Vorschlag: 7835 [↵]
0 Bullen; 4 Kuehe

Dein Vorschlag: 8573 [↵]
1 Bullen; 3 Kuehe

Dein Vorschlag: 5783 [↵]
```

4 Bullen; 0 Kuehe

Du hast die Zahl nach 7 Versuchen gefunden

\$

## 5.28.2 Realisierung des Kommandos cat

Das folgende Beispiel zeigt eine mögliche Realisierung des Kommandos **cat**.

```
$ cat mcat
#-----
ksh-Version von cat
#
urspruengliche Version aus:
The KornShell,
Morris I. Bolsky, David G.Korn
Ein dort enthaltener Fehler ist hier
behoben.
#-----
alias open=exec # zwecks besserer Lesbarkeit
alias duplicate=exec # zwecks besserer Lesbarkeit
duplicate 3<&0 # Dateideskr. 0 sichern

if (($# == 0))
then set - - # - steht fuer Standardeingabe
fi

for i # fuer jede Datei
do if test x"$i" != x # Datei eroeffnen, wenn noetig
 then if test x"$i" != x- # - ist Standardeingabe
 then open 0< $i
 else duplicate 0<&3 # verwende Stdeingabe
 fi
 fi
 IFS=
 while read -r zeile
 do print -r - "$zeile"
 done
done
$ cx mcat
$ mcat
das ist ein Test
das ist ein Test
Hoffentlich erfolgreich
Hoffentlich erfolgreich
Strg-D
$
```

### 5.28.3 Banner-Ausgabe im Querformat mit qbanner

Während das Kommando **banner** von BSD-Unix den in Spruchbandform auszugebenden Text bereits im Querformat ausgibt, um ihn dann auch auf Endlospapier drucken zu lassen, ist dies beim **banner** von System V nicht der Fall. Mit dem nachfolgenden Skript **qbanner** kann auch auf System V eine Querformatausgabe erreicht werden:

```
$ cat qbanner ↵
#-----
qbanner - banner mit Querformat-Ausgabe
#
#-----

#--- Cleanup bei Fehler und Ende
#-----
trap 'if [[-f $tmpdat]]
then rm $tmpdat
fi' ERR EXIT

#--- Mindestens ein Argument vorhanden ?
#-----
if [$# -eq 0]
then
echo "usage: $0 argument(e)"
exit 1
fi

#--- Deklarationen und Initialisierung
#-----
integer i=1 j z=1 l laenge[511] m max
typeset -L1 zeich
tmpdat="/tmp/qbanner.$$"

#--- Banner in temporäre Datei; Dabei werden alle
#--- alle Leerz. in Unterstrich umgewandelt und alle
#--- Nummernzeichen in X umgewandelt, um IFS nicht
#--- nicht explizit anpassen zu müssen.
#-----
banner "$@" | tr " # " "_X" >$tmpdat

#--- Zeile fuer Zeile aus temporären Datei lesen
#-----
cat $tmpdat |
while read
do zeile[i]="$REPLY" # im Array zeile speichern
if ((i%8==0)) # eine Banner-Zeile
= 8 normale Zeilen
then max=0
j=i-7
while ((j<=i)) # maximale Laenge in
do zeil="{zeile[j]}" # Banner-Zeile
```

```

 laenge[j]={#zeil} # bestimmen
 if ((laenge[j]>max))
 then max=laenge[j]
 fi
 j=j+1
 done
 j=i-7
 while ((j<=i)) # alle Banner-Zeilen
 do m=laenge[j] # gleich lang machen
 while ((m<=max))
 do zeile[j]="${zeile[j]}_"
 m=m+1
 done
 j=j+1
 done
fi
i=i+1
done

i=i-1 # i enthaelt die Zeilenzahl

#--- Ausgeben der Banner-Zeilen im Querformat
#-----
while ((z<=i))
do
 l=1
 while ((l<=78))
 do j=z+7
 ausgabe=""
 while ((j>=z))
 do x="${zeile[j]}"
 if [[-n "$x"]]
 then
 zeich="$x"
 if [["$zeich" = "_"]]
 then ausgabe="$ausgabe "
 else ausgabe="${ausgabe}XXXXX"
 fi
 zeile[j]="${x#${zeich}}"
 fi
 j=j-1
 done
 if [[-n "$x"]]
 then echo "$ausgabe\n$ausgabe\n$ausgabe"
 fi
 l=l+1
 done
 echo "\n\n\n\n"
 z=z+8
done

$ cx qbanner
$ qbanner Happy Birthday

```

[illegible]



[illegible][illegible]

```
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXX
XXXXXX
XXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXX
XXXXXX
XXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
```

• • • • •

• • • • •

• • • • •

• • • • •

\$

Mit

**qbanner text | lp**

könnte die **qbanner**-Ausgabe direkt an den Drucker weitergeleitet werden.

#### 5.28.4 Zeilenweiser Vergleich zweier Dateien mit mdiff

Das nachfolgende Skript **mdiff** macht einen zeilenweisen Vergleich von zwei Dateien. Sind zwei Zeilen gleich, so schreibt dieses Skript die entsprechende Zeile mit Zeilennummer in die Datei *passt*, andernfalls werden die beiden unterschiedlichen Zeilen in die Datei *anders* geschrieben. Dieses Skript soll vor allen Dingen zeigen, wie in einem Skript gleichzeitig mehrere Dateien zum Lesen und Schreiben eröffnet werden können:

```
$ cat mdiff ↵
#-----
mdiff vergleicht 2 Dateien zeilenweise:
mdiff datei1 datei2
#-----

#.... zwecks besserer Lesbarkeit
alias open=exec

#.... 2 Argumente muessen angegeben sein
if (($# != 2))
```

```
then echo "usage: $0 datei1 datei2" >&2
 exit 1
fi

#... Beide Argumente muessen lesbare Dateien sein
if [[! ((-f $1) && (-r $1))]]
then echo "Fehler: $1 ist keine lesbare Datei" >&2
 exit 1
fi
if [[! ((-f $2) && (-r $2))]]
then echo "Fehler: $2 ist keine lesbare Datei" >&2
 exit 1
fi

#.... 4 Dateien eroeffnen
open 3< $1 # $1 zum Lesen eroeffnen
open 4< $2 # $2 zum Lesen eroeffnen
open 5> passt # Datei "passt" zum Schreiben eroeffnen
open 6> anders # Datei "anders" zum Schreiben eroeffnen

#.... $1 und $2 zeilenweise vergleichen
integer pid1=0 pid2=0
typeset -R5 nr=1
IFS=
while true
do
 read -u3 -r zeil1 # Zeile von $1 lesen
 pid1=$? # exit-Status von read
 read -u4 -r zeil2 # Zeile von $2 lesen
 pid2=$? # exit-Status von read
 if (((pid1 != 0) && (pid2 != 0)))
 then break
 fi
 if ["$zeil1" = "$zeil2"] # Zeilen vergleichen
 then print -u5 -R "$nr:$zeil1"
 else print -u6 -R "$nr:$zeil1 <-> $zeil2"
 fi
 ((nr=nr+1))
done

#.... Meldung, dass fertig
echo "Gleiche Zeilen in Datei 'passt'" >&2
echo "Unterschiedliche Zeilen in Datei 'anders'" >&2
$ cx mdiff ↵
$
```

### 5.28.5 Realisierung des Kommandos head

Auf vielen Systemen wird ein Kommando **head** angeboten. Mit dem Kommando **head** können die ersten Zeilen einer Datei ausgegeben werden. **head** ist somit das Gegenstück zum Kommando **tail**. Die Aufrufsyntax für **head** ist üblicherweise

**head** [-n] [datei(en)]

Die Zahl *n* legt dabei fest, wie viele Zeilen von den angegebenen *datei(en)* jeweils auszugeben sind. Ist *-n* nicht angegeben, so ist die Voreinstellung 10 Zeilen. Nachfolgend wird eine mögliche Realisierung dieses Kommandos **head** gegeben:

```
$ cat head ↵
#-----
head - Die ersten Zeilen von Dateien ausgeben
#
head [-n] [datei(en)]
-n die Zahl n legt die Anzahl
der auszugebenden Zeilen fest.
Voreinstellung ist: 10
#-----

#... Initialisierungen
alias open=exec # zwecks besserer Lesbarkeit
alias duplicate=exec # zwecks besserer Lesbarkeit
duplicate 3<&0 # Dateideskr. 0 sichern

#... auszugebende Zeilenzahl festlegen
typeset -l1 zeich="$1"
integer n zahl=10
if [["$zeich" = "-"]]
then zahl=${1#"-"}
 shift
fi

#... wenn keine Dateien angegeb., dann Stdeingabe
if (($# == 0))
then set - - # - steht fuer Standardeingabe
fi

#... Dateien der Reihe nach ausgeben
for i
do if test x"$i" != x # Datei eroeffnen, wenn noetig
 then if test x"$i" != x- # - ist Standardeingabe
 then open 0< $i
 if (($# > 1)) # bei mehreren Dateien
 then echo "==> $i <==" # Dateiname als
 fi # erstes ausgeben.
 else duplicate 0<&3 # verwende Stdeing.
```

```

 fi
 fi
 IFS=
 n=zahl # n = Zahl der auszugebenden Zeilen
 while ((n=n-1) >= 0) && read -r zeile
 do print -r - "$zeile"
 done
done
$ cx head
$./head -15 head [ersten 15 Zeilen der Datei head ausgeben]
#-----
head - Die ersten Zeilen von Dateien ausgeben
#
head [-n] [datei(en)]
-n die Zahl n legt die Anzahl
der auszugebenden Zeilen fest.
Voreinstellung ist: 10
#-----

#... Initialisierungen
alias open=exec # zwecks besserer Lesbarkeit
alias duplicate=exec # zwecks besserer Lesbarkeit
duplicate 3<&0 # Dateideskr. 0 sichern

#... auszugebende Zeilenzahl festlegen
$

```

## 5.28.6 Dateien eines Directorybaums nach Größe auflisten (dusort)

Das nachfolgende Skript **dusort** gibt die Größe aller Dateien von den beim Aufruf angegebenen Directorybäumen aus. Die Dateien werden dabei bei der Ausgabe der Größe nach sortiert. Ist die Option **-r** nicht angegeben, so wird die kleinste (ansonsten die größte) Datei zuerst ausgegeben:

```

$ cat dusort
#-----
dusort: gibt die Groesse aller Dateien von
Directorybaeumen (im du-Format) aus.
Dateien werden dabei nach Groesse sortiert.
#
Syntax:
dusort [-r] pfadname(n)
-r Groesste zuerst;
Voreinstellung: Kleinste zuerst
#-----

#--- Name fuer temporaere Datei ---
#-----
tmp_datei="/tmp/dusort.$$"

```

```

#--- Signalhandler einrichten -----
#-----
trap 'rm $tmp_datei 2>/dev/null; exit 1' INT QUIT EXIT

#--- Funktion baum -----
#-----
function baum
{
 cd $1
 set +o noglob # Dateinamen-Expandierung einschalten
 typeset datei wd_dateien="$(echo .* *)" # Lokale Var.

 set -o noglob # Dateinamen-Expandierung ausschalten,
 # um zweimalige Auswertung von
 # Sonderzeichen in Dateinamen
 # zu verhindern.
 for datei in $wd_dateien
 do if [[$datei -ef . || $datei -ef ..]]
 then continue # Fuer . und .. nichts tun
 fi
 if [[-d $datei]] # Wenn Directory dann
 then baum $datei # baum rekursiv aufrufen
 cd ..
 elif [[-r $datei]]
 then du -s $PWD/$datei >>$tmp_datei
 fi
 done
 set +o noglob # Dateinamen-Expandierung einschalten
}

#--- main -----
#-----

#... Optionen abarbeiten
while getopts r opt
do case $opt in
 r) option="-r";;
 \?) echo "usage: $0 [-r] pfadname(n)"
 exit 1;;
 esac
done
integer anz=OPTIND-1
shift $anz

#... baum fuer alle angegebenen Pfade aufrufen ...
startdir="$PWD"
for pfad
do baum $pfad
 cd $startdir # falls rel. Pfadnamen in Kdozeile
done

#... Inhalt von $tmp_datei nach Groesse
#... sortiert ausgeben

```

```
if [[-s $tmp_datei]]
then sort $option -n $tmp_datei
fi
$ cx dusort
$ dusort -r $HOME
74 /user1/egon/kshueb/a.out
58 /user1/egon/kshueb/core
19 /user1/egon/kshueb/stdio.h
12 /user1/egon/.sh_history
5 /user1/egon/kshueb/dusort
4 /user1/egon/kshueb/hanoi
3 /user1/egon/.ksh_env
2 /user1/egon/mbox
2 /user1/egon/kshueb/vonhinten
2 /user1/egon/kshueb/settext
2 /user1/egon/kshueb/datana1
1 /user1/egon/kshueb/zufall
1 /user1/egon/kshueb/zaehle.txt
1 /user1/egon/kshueb/wartmsg
: :
: :
: :
1 /user1/egon/flib/_from
1 /user1/egon/.profile
0 /user1/egon/kshueb/neudat
0 /user1/egon/kshueb/abc2
0 /user1/egon/kshueb/abc
$
```

## 5.28.7 Eine Adressenverwaltung (adr)

Das umfangreiche Skript **adr** realisiert eine einfache Adressenverwaltung.

Werden beim Aufruf dieses Skripts Argumente angegeben, so wird im Adreßbuch nach Einträgen gesucht, in denen die angegebenen Strings vorkommen. Werden Einträge gefunden, so werden dazu die entsprechenden Adressen ausgegeben.

Wird **adr** ohne jegliche Argumente aufgerufen, so arbeitet es interaktiv. Über ein Menü kann dann der Benutzer die gewünschte Aktion anwählen. Zur Auswahl stehen dabei:

- ▶ Adresse suchen
- ▶ Adresse hinzufügen
- ▶ Adresse löschen
- ▶ Adresse ändern
- ▶ Alle Adressen auflisten

```

$ cat adr ↵
#-----
adr - Ein Programm zur Adressenverwaltung
Es bietet folgende Funktionen an:
Adresse suchen
Adresse hinzufuegen
Adresse entfernen
Adresse aendern
Alle Adressen auflisten
#
Syntax:
adr [argument(e)]
Sind argument(e) angegeben, so werden
Zeilen im Adressenbuch gesucht, die
diese Argumente enthalten.
Sind keine argument(e) angegeben,
so kann der Benutzer ueber ein
Menue die geforderten Funktionen
abrufen.
#-----

#.... Deklarationen und Initialisierungen
typeset -l -L1 antwort
alias open=exec
adrbuch="$HOME/adressbuch"

#.... Funktion ausgabe
gibt einen Eintrag aus dem Telefonbuch aus.
#.....
function ausgabe
{
 typeset i # Lokale Variable
 echo "\n+-----+"
 IFS="@"
 set - $1
 for i
 do echo " |\r| $i"
 done
 echo "+-----+"
}

#.... Funktion suche
sucht Eintraege im Telefonbuch und gibt
gefundene Eintraege aus.
#.....
function suche
{
 typeset tmpdat="/tmp/gefunden.$$"
 typeset muster

 trap 'rm -r $tmpdat 2>/dev/null' EXIT INT QUIT

```



```
if (($# == 0))
then read muster?"Muster fuer die zu suchenden Eintraege: "
else muster="$1"
fi
while [[-n "$muster"]]
do grep "$muster" $adrbuch >$tmpdat
 if [[-s $tmpdat]]
 then open 3< $tmpdat
 while read -u3 -r zeile
 do ausgabe "$zeile"
 done
 else echo "\007Muster '$muster' nicht gefunden" >&2
 fi
 if (($# != 0))
 then break
 fi
 read muster?"Muster fuer die zu suchenden Eintraege: "
done
}

#.... Funktion add
fuegt weitere Eintraege ins Adressbuch ein
#.....
function add
{
 typeset name zusatz strasse ort telnr
 trap 'sort -o $adrbuch $adrbuch' EXIT INT QUIT

 read name?"Name/Firmaname: "
 while [[-n "$name"]]
 do read zusatz?"Zusaetzliche Bezeichnungen: "
 read strasse?"Strasse: "
 read ort?"Wohnort: "
 read telnr?"Telefonnummer: "
 echo "$name@$zusatz@$strasse@$ort@$telnr" >>$adrbuch
 read name?"Name/Firmaname: "
 done
}

#.... Funktion loesch
loescht Eintraege im Adressbuch.
#.....
function loesch
{
 typeset tmpdat="/tmp/loesch.$$"
 typeset zwidat="/tmp/zwischdat.$$"
 typeset muster

 trap 'rm -r $tmpdat 2>/dev/null
 rm -r $zwidat 2>/dev/null' EXIT INT QUIT
```

```

read muster?"Muster fuer die zu loeschenden Eintraege: "
while [[-n "$muster"]]
do grep "$muster" $adrbuch >$tmpdat
 if [[-s $tmpdat]]
 then open 3< $tmpdat
 while read -u3 -r zeile
 do ausgabe "$zeile"
 read antwort?"Diesen Eintrag loeschen (j/n) ?"
 if [["$antwort" = "j"]]
 then grep -v "^$zeile$" $adrbuch >$zwidat
 mv $zwidat $adrbuch
 fi
 done
 else echo "\007Muster '$muster' nicht gefunden" >&2
 fi
 read muster?"Muster fuer die zu loeschenden Eintraege: "
done
}

#.... Funktion aendern
aendert Eintraege im Adressbuch.
#.....
function aendern
{
 typeset tmpdat="/tmp/loesch.$$"
 typeset zwidat="/tmp/zwischdat.$$"
 typeset edidat="/tmp/edidat.$$"
 typeset editor=${EDITOR:-vi}
 typeset muster

 trap 'rm -r $tmpdat 2>/dev/null
 rm -r $zwidat 2>/dev/null
 rm -r $edidat 2>/dev/null' EXIT INT QUIT

 read muster?"Muster fuer die zu aendernden Eintraege: "
 while [[-n "$muster"]]
 do grep "$muster" $adrbuch >$tmpdat
 if [[-s $tmpdat]]
 then open 3< $tmpdat
 while read -u3 -r zeile
 do ausgabe "$zeile"
 read antwort?"Diesen Eintrag aendern (j/n) ?"
 if [["$antwort" = "j"]]
 then echo -n "$zeile" |
 tr "@" "\012" >$edidat
 echo -n "\007Aenderungen mit dem " >&2
 echo "Editor $editor durchfuehren" >&2
 read dummy?"Weiter mit Return....."
 $editor $edidat
 grep -v "^$zeile$" $adrbuch >$zwidat
 tr "\012" "@" < $edidat >>$zwidat

```

```

 echo >> $zwidat # fehlendes Neuezeile-
 # Zeichen anhaengen
 sort -o $zwidat $zwidat
 mv $zwidat $adrbuch
 fi
done
else echo "\007Muster '$muster' nicht gefunden" >&2
fi
read muster?"Muster fuer die zu aendernden Eintraege: "
done
}

#.... Funktion listealle
gibt alle Eintraege des Adressbuchs aus.
#.....
function listealle
{
 integer einmal=0 # fuer die Ausgabe der Adressabtrennung
 typeset zeile
 typeset tel="$1" # $1 leer --> Adressen, sonst
 # Telefonnummern
 typeset -L20 name zusatz leer=" "
 typeset -L20 strasse ort
 typeset -R30 telnr
 open 3< $adrbuch
 { echo "\n-----"
 IFS="@ "
 while read -u3 -r zeile
 do set - $zeile
 name="$1"
 if [[-z "$tel"]]
 then if ((einmal))
 then
 echo " - - - - -"
 fi
 einmal=1
 zusatz="$2"
 strasse="$3"
 ort="$4"
 echo "$name$zusatz\n$leer$strasse\n$leer$ort"
 else
 telnr="$5"
 echo "$name$telnr"
 fi
 done
 echo "-----"; } |
 pg -e
}

#.... main
#.....

```

```
if [[! -f "$adrbuch"]]
then echo "Adressbuch '$adrbuch' existiert nicht" >&2
read antwort?"Soll eines erstellt werden (j/n) ?"
if [["$antwort" != "j"]]
then exit 1
fi
>$adrbuch ||
{ echo -n "\007Anlegen von Adressbuch '$adrbuch' " >&2
echo "nicht moeglich" >&2
exit 1
}
fi

if (($# != 0))
then for i
do suche "$i"
done
exit 0
fi

PS3="Ihre Wahl ? "
select i in "Adresse suchen" \
"Adresse hinzufuegen" \
"Adresse loeschen" \
"Adresse veraendern" \
"Alle Adressen auflisten" \
"Alle Telefonnummer auflisten" \
"Adress-Verwaltung beenden"
do
case "$REPLY" in
1) suche;;
2) add;;
3) loesch;;
4) aendern;;
5) listealle;;
6) listealle "x";;
7) exit 0;;
*) echo "\007Falsche Auswahl" >&2;;
esac
done
$ cx adr ↵
$ adr ↵
Adressbuch '/user1/egon/adressbuch' existiert nicht
Soll eines erstellt werden (j/n) ?j ↵
1) Adresse suchen
2) Adresse hinzufuegen
3) Adresse loeschen
4) Adresse veraendern
5) Alle Adressen auflisten
6) Alle Telefonnummer auflisten
7) Adress-Verwaltung beenden
```

```
Ihre Wahl ? 2
Name/Firmaname: Mueller Toni
Zusaetzliche Bezeichnungen: Malerbetrieb
Strasse: Talstr. 176
Wohnort: 91056 Erlangen
Telefonnummer: 09131/1234567
Name/Firmaname: Aller Gunther
Zusaetzliche Bezeichnungen: SW-Buero
Strasse: Eckenweg 123
Wohnort: 90480 Nuernberg
Telefonnummer: 0911/8765432
Name/Firmaname:
Ihre Wahl ?
1) Adresse suchen
2) Adresse hinzufuegen
3) Adresse loeschen
4) Adresse veraendern
5) Alle Adressen auflisten
6) Alle Telefonnummer auflisten
7) Adress-Verwaltung beenden
Ihre Wahl ? 6

Aller Gunther 0911/8765432
Mueller Toni 09131/1234567

Ihre Wahl ? 2
Name/Firmaname: Beller Franziska
Zusaetzliche Bezeichnungen: SW-Beratung
Strasse: Bergstr. 444
Wohnort: 97076 Wuerzburg
Telefonnummer: 0977/152556
Name/Firmaname: Stuermer Sophie
Zusaetzliche Bezeichnungen: SW-Entwicklung
Strasse: Zeppelinstr. 16
Wohnort: 97535 Gressthal
Telefonnummer: 09726/1234
Name/Firmaname:
Ihre Wahl ?
1) Adresse suchen
2) Adresse hinzufuegen
3) Adresse loeschen
4) Adresse veraendern
5) Alle Adressen auflisten
6) Alle Telefonnummer auflisten
7) Adress-Verwaltung beenden
Ihre Wahl ? 5

Aller Gunther SW-Buero
 Eckenweg 123
```

|                  |                                                      |
|------------------|------------------------------------------------------|
|                  | 90480 Nuernberg                                      |
| Beller Franziska | SW-Beratung<br>Bergstr. 444<br>97076 Wuerzburg       |
| Mueller Toni     | Malerbetrieb<br>Talstr. 176<br>91056 Erlangen        |
| Stuermer Sophie  | SW-Entwicklung<br>Zeppelinstr. 16<br>97535 Gressthal |

Ihre Wahl ? 2

Name/Firmaname:

Ihre Wahl ? 

- 1) Adresse suchen
- 2) Adresse hinzufuegen
- 3) Adresse loeschen
- 4) Adresse veraendern
- 5) Alle Adressen auflisten
- 6) Alle Telefonnummer auflisten
- 7) Adress-Verwaltung beenden

Ihre Wahl ? 1 

Muster für die zu suchenden Einträge: 091

Aller Gunther  
 SW-Buero  
 Eckenweg 123  
 90480 Nuernberg  
 0911/8765432

-----+-----  
Mueller Toni  
Malerbetrieb  
Talstr. 176  
91056 Erlangen  
09131/1234567  
-----+-----

Muster für die zu suchenden Einträge:

Ihre Wahl ?

- 1) Adresse suchen
- 2) Adresse hinzufuegen
- 3) Adresse loeschen
- 4) Adresse veraendern
- 5) Alle Adressen auflisten
- 6) Alle Telefonnummer auflisten

7) Adress-Verwaltung beenden

Ihre Wahl ? 7

\$ adr Stuerm

```
+-----+
| Stuermer Sophie |
| SW-Entwicklung |
| Zeppelinstr. 16 |
| 97535 Gressthal |
| 09726/1234 |
+-----+
```

\$ adr 09131

```
+-----+
| Mueller Toni |
| Malerbetrieb |
| Talstr. 176 |
| 91056 Erlangen |
| 09131/1234567 |
+-----+
```

\$

## 5.29 Unterschiede zwischen der Bourne-Shell und ksh-Versionen

Dieses Kapitel faßt die wichtigsten Unterschiede zusammen, die zwischen

- ▶ der ursprünglichen Bourne-Shell,
- ▶ der Bourne-Shell, die mit System V Release 3 freigegeben wurde,
- ▶ der **ksh**-Version vom 3.6.1986 und
- ▶ der **ksh**-Version vom 16.11.1988

bestehen.

### 5.29.1 ksh-Konstrukte, die nicht in der ursprünglichen Bourne-Shell vorhanden sind

- ▶ Kommentar-Kommando # (dafür mußte Kommando : verwendet werden)
- ▶ Metazeichen [!..] für Dateinamen-Expandierung
- ▶ : (Doppelpunkt) bei den speziellen Variablenausdrücken
- ▶ Ein-/Ausgabeumlenkung bei builtin-Kommandos
- ▶ Shell-Funktionen
- ▶ **test** und **echo** waren keine builtin-Kommandos

### 5.29.2 ksh-Konstrukte, die nicht in der System-V-Bourne-Shell vorhanden sind

Die folgenden Konstrukte sind in der **ksh**-Version vom 16.11.1988 vorhanden, aber nicht in der Bourne-Shell, die mit System V Release 3 freigegeben wurde:

- ▶ Variablen-Attribute
  - Arrays
  - eingebaute Arithmetik
  - Aliase
  - Tilde-Mechanismus
  - Ko-Prozesse
- ▶ Operatoren **((.))** | **&** **>** | **<>**
- ▶ Schlüsselwörter **function**, **time**, **select**, **[[** und **]]**
- ▶ die speziellen Variablen-Ausdrücke **\${variable#pattern}**, **\${variable##pattern}**, **\${variable%pattern}**, **\${variable%%pattern}** und **\${#variable}**.
- ▶ Positionsparameter größer als 9 (**\${10}**, **\${11}**, ..)
- ▶ die Kommandosubstitution **\$(kdo liste)**.
- ▶ Dateinamen-Expandierung mit **\*(pattern)**, **?(pattern)**, **+(pattern)**, **@(pattern)** und **!(pattern)**.
- ▶ die Variablen **\_**, **ENV**, **ERRNO**, **FCEDIT**, **FPATH**, **HISTFILE**, **HISTSIZE**, **LINENO**, **OLDPWD**, **PPID**, **PS3**, **PS4**, **PWD**, **RANDOM**, **REPLY**, **SECONDS** und **TMOU**.
- ▶ die builtin-Kommandos **alias**, **bg**, **fc**, **fg**, **jobs**, **let**, **print**, **typeset**, **unalias** und **whence**.
- ▶ **test**-Operatoren **-nt**, **-ot**, **-ef**, **-O**, **-G** und **-S**.
- ▶ **cd** – und der Aufruf von **cd** mit zwei Argumenten.
- ▶ Zuweisen von Werten bei den builtin-Kommandos **readonly** und **export**.
- ▶ Symbolische Namen für Signale.
- ▶ Namen für Optionen und **set -o optname**.

### 5.29.3 Unterschiede zwischen der ksh und der System-V-Bourne-Shell

Im folgenden werden die wesentlichen Unterschiede zwischen der **ksh**-Version vom 16.11.1988 und der Bourne-Shell von System V Release 3 aufgezeigt. Die Bourne-Shell von System V Release 3 wird dabei mit **sh** bezeichnet. Auf die Unterschiede der einzelnen builtin-Kommandos wird dabei nicht eingegangen. Diese sind im Anhang bei den jeweiligen builtin-Kommandos angegeben.



**Zeichen ^**

**sh** Alternative Angabe für das Pipeline-Zeichen |.

**ksh** Keine Sonderbedeutung.

**Variable IFS**

**sh** Darin enthaltene Trennzeichen gelten für alle Wörter; z.B. würde  
IFS=i; print in folgendem Aufruf resultieren:  
pr nt

**ksh** Darin enthaltene Trennzeichen gelten nur für **read**, **set** und für das Resultat aus Parameter- und Kommandosubstitution.

**Modifizieren der Umgebungsvariablen**

**sh** Modifizierte Variablen müssen explizit für den Export markiert werden, damit deren Wert an einen Kindprozeß vererbt wird.

**ksh** Der Wert von modifizierten Variablen wird automatisch an Kindprozesse vererbt.

**time**

**sh** Kein Schlüsselwort. Auch können mit **time** keine Zeitmessungen für builtin-Kommandos oder Funktionen durchgeführt werden.

**ksh** Schlüsselwort. **time x | y** mißt die von dieser Pipeline benötigte CPU-Zeit; in der **sh** würde nur die Zeit für **x** gemessen.

**select und function**

**sh** Keine Schlüsselwörter.

**ksh** Schlüsselwörter.

**for, while, until mit Ein/Ausgaben-Umlenkung**

**sh** Werden in einer eigenen Prozeß-Umgebung ausgeführt.

**ksh** Werden in der momentanen Prozeß-Umgebung ausgeführt.  
Zuweisungen, die innerhalb dieser Schleifen vorgenommen wurden, gelten auch noch nach Beendigung der Schleife.

### Funktionen

- sh**
- Alle Variablen einer Funktion sind global.
  - Fehler in einer Funktion bewirken das Ausführungsende des aufrufenden Skripts.
  - Positionsparameter **0** wird durch einen Funktionsaufruf nicht neu gesetzt.
  - Signal-Handler einer Funktion sind global.
  - Funktionen und Variablen dürfen keine gleichen Namen besitzen.
- ksh**
- Funktionslokale Variablen sind möglich.
  - Fehler in einer Funktion bewirken nur das Ausführungsende der betreffenden Funktion und nicht des aufrufenden Skripts.
  - Positionsparameter **0** wird beim Funktionsaufruf mit dem Funktionsnamen besetzt.
  - Signal-Handler, die in einer Funktion installiert werden, sind lokal.
  - Funktionen und Variablen dürfen gleiche Namen besitzen; beim Löschen mit **unset** muß dann **-f** angegeben werden, wenn die Funktion entfernt werden soll.

### Tilde-Expandierung für Wörter, die mit ~ beginnen

- sh** Nicht möglich.
- ksh** Möglich.

### ((..)) anstelle eines Kommandonamens

- sh** Bedeutet geschachtelte Kommandoklammerung mit (..)
- ksh** Arithmetischer Ausdruck.

### Zwei aufeinanderfolgende IFS-Trennzeichen (nicht Leer- oder Tabulatorzeichen)

- sh** Werden als ein Trennzeichen gewertet.
- ksh** Wird als leeres Argument gewertet.

### Vergleichsoperatoren bei test (-gt, -lt, -ne, ...)

- sh** Operanden müssen Konstanten sein.
- ksh** Operanden können beliebige arithmetische Ausdrücke sein.

### hash

- sh** builtin-Kommando.

**ksh** Vordefiniertes Alias; bis auf die Option **-r** weitgehend ähnlich zum builtin-Kommando **hash** der **sh**. Um **hash -r** in **ksh** nachzubilden, muß **PATH=\$PATH** eingegeben werden.

#### **set -- ohne Argumente**

**sh** Keine Auswirkung.

**ksh** Löscht die Positionsparameter.

#### **Fehler bei Ausführung eines builtin-Kommandos**

**sh** Bewirkt immer den Abbruch des entsprechenden Skripts.

**ksh** Bewirkt bis auf wenige Ausnahmen nicht den Abbruch des Skripts.

#### **Kommandosubstitution mit \$(..)**

**sh** Nicht möglich.

**ksh** Möglich.

**reale UID nicht gleich der effektiven UID bzw.**

**reale GID nicht gleich der effektiven GID**

**sh** Keine Änderungen durch **sh**.

**ksh** Setzt die effektive UID bzw. GID zurück auf die reale UID bzw. GID, wenn die Option **privileged** nicht eingeschaltet ist.

### **5.29.4 Neuheiten der ksh-Version vom 16.11.1988 gegenüber früheren ksh-Versionen**

#### **Neue Möglichkeiten bei Dateinamen-Expandierung**

**?(pattern[ | pattern]...)** Deckt kein oder ein Vorkommen der angegebenen Pattern ab.

**\*(pattern[ | pattern]...)** Deckt kein, ein oder mehrere Vorkommen der angegebenen Pattern ab.

**+(pattern[ | pattern]...)** Deckt ein oder mehrere Vorkommen der angegebenen Pattern ab.

**@(pattern[ | pattern]...)** Deckt genau ein Vorkommen der angegebenen Pattern ab.

**!(pattern[ | pattern]...)** Deckt die Strings ab, die durch keines der angegebenen Pattern abgedeckt werden

### Neue Alternative zum Kommando test

Das Kommando **[[..]]** macht das builtin-Kommando **test** (bzw. **[..]**) überflüssig. Die dabei neu eingeführten Operatoren **&&** und **||** ersetzen die Operatoren **-a** und **-o**.

Für den in **[[..]]** angegebenen Text wird keine Aufteilung in einzelne Wörter und auch keine Dateinamen-Expandierung durchgeführt.

Bei den Operatoren **=** und **!=** darf als rechter Operand ein pattern angegeben werden.

Zum Vergleich zweier Strings dürfen auch die Operatoren **<** und **>** verwendet werden.

### Neue arithmetische Operatoren

Die neuen arithmetischen Operatoren **|**, **&**, **||**, **&&**, **^**, **<<**, **>>** und **~** dürfen in jedem beliebigen arithmetischen Ausdruck verwendet werden.

Auch sind die aus C bekannten zusammengesetzten Operatoren **+=**, **-=**, ... neu in dieser **ksh**-Version hinzugekommen.

Für alle arithmetischen Operatoren gelten dabei die gleichen Prioritätsregeln wie in C.

### Neue Shell-Variablen

|               |                                                                                                                                                                                                                                                                                                                                |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ERRNO</b>  | Wird nach jedem fehlerhaften Systemaufruf auf die entsprechende Fehlernummer gesetzt.                                                                                                                                                                                                                                          |
| <b>FPATH</b>  | Enthält Directories, welche nach Funktionsdefinitionen zu durchsuchen sind. Mit der Deklaration<br><b>typeset -fu funktionsname</b><br>wird festgelegt, daß <i>funktionsname</i> in den <b>FPATH</b> -Directories zu suchen ist.<br>Ein vordefiniertes Alias<br><b>autoload='typeset -fu'</b><br>wurde ebenso neu hinzugefügt. |
| <b>LINENO</b> | Wird auf die Zeilennummer des aktuellen Kommandos in einem Shell-Skript gesetzt.                                                                                                                                                                                                                                               |
| <b>PS4</b>    | Enthält den Promptstring, der beim Debuggen (Option <b>-x</b> ) auszugeben ist.                                                                                                                                                                                                                                                |
| <b>OPTARG</b> | Wird vom Kommando <b>getopts</b> gesetzt ( <b>getopts</b> ist in der System-V-Shell auch vorhanden)                                                                                                                                                                                                                            |
| <b>OPTIND</b> | Wird vom Kommando <b>getopts</b> gesetzt.                                                                                                                                                                                                                                                                                      |

### Änderungen bei Shell-Variablen

Die Attribute für Variablen, die für den Export markiert sind, werden bei neuen **ksh**-Aufrufen übernommen. So wird zum Beispiel eine zum Export markierte Variable, die zusätzlich noch das `readonly`-Attribut gesetzt hat, bei jedem neuen Aufruf einer **ksh** auch dort nur lesbar sein.

Die Shell-Variable **IFS** wird immer nach dem Lesen der Environment-Datei auf ihren voreingestellten Wert zurückgesetzt, selbst wenn sie exportiert wurde.

### Neuheiten und Änderungen bei den builtin-Editoren

- ▶ Suchen mit `^string` deckt nur *strings* ab, die am Anfang einer Zeile stehen; dies gilt für **emacs** und **vi**.
- ▶ Die **vi**-Direktive `\` bewirkt, daß das momentane Cursor-Wort so weit expandiert wird, wie es einen eindeutigen Pfadnamen abdeckt.
- ▶ Die **emacs**-Direktive `ESC ESC` entspricht der **vi**-Direktive `\`.
- ▶ Die **vi**-Direktive `|` ist neu hinzugekommen.
- ▶ Vor der **vi**-Direktive `r` kann eine Zahl *n* angegeben werden, um *n* Zeichen ab Cursorposition zu ersetzen.

### Änderungen bei builtin-Kommandos

- ▶ Die Escape-Sequenz `\a` (Klingelton) ist nun bei dem builtin-Kommando **print** bzw. **echo** erlaubt.
- ▶ Neue Optionen bei **test**, `[..]` und `[[..]]` sind **-O**, **-G** und **-S**.
- ▶ Es können Array-Variablen mit **set -A name wert** Werte zugewiesen werden.
- ▶ Das neu hinzugekommene Signal **DEBUG** wird nach jeder Ausführung eines Kommandos geschickt.
- ▶ **typeset -p** ist nicht mehr möglich.
- ▶ Neu hinzugekommen ist das builtin-Kommando **getopts**.

### Neue Optionen für builtin-Kommandos

**set**      Option **nolog**  
          Option **noclobber**  
          Option **-p** bedeutet nun **privileged**  
          **set -o bgnice** ist die Voreinstellung

**typeset** Neu ist die Deklarations-Möglichkeit  
**typeset -fu** *funktionsname*  
Damit wird festgelegt, daß *funktionsname* in den **FPATH**-Directories zu suchen ist.  
Ein vordefiniertes Alias  
**autoload='typeset -fu'**  
wurde ebenso neu hinzugefügt.

### Änderungen bei Ko-Prozessen

Es ist nun möglich, Dateideskriptoren für einen Ko-Prozeß (Job gefolgt von | &) auf numerierte Dateideskriptoren umzulenken. Um den lesenden Deskriptor umzulenken, muß **exec n<&p**, um den schreibenden Deskriptor umzulenken, muß **exec n>&p** angegeben werden; *n* muß dabei die Nummer des entsprechenden Dateideskriptors sein. Nach einer solchen Umlenkung ist es dann möglich, einen solchen Dateideskriptor zu schließen oder die Eingabe bzw. Ausgabe eines anderen Kommandos dort hineinzuleiten.

### Änderungen bei der Job-Kontrolle

Neu ist, daß mit *%?string* ein Job angesprochen werden kann, bei dem *string* in seiner Kommandozeile vorkommt.  
Beim builtin-Kommando **wait** können nicht nur mehr PIDs, sondern auch Job-Kennungen angegeben werden.

### Sonstige Neuheiten und Änderungen

- ▶ Variablen-Zuweisungen werden nun von links nach rechts ausgewertet.
- ▶ Jedem pattern in einer **case**-Anweisung kann nun eine öffnende Klammer vorangestellt werden. Diese ( muß sogar angegeben werden, wenn sich die betreffende **case**-Anweisung innerhalb einer Kommandosubstitution **\$(..)** befindet.
- ▶ Der Umlenkungs-Operator **<>** löscht nicht mehr zuerst den vollständigen Inhalt der entsprechenden Datei, bevor er diese zum Lesen und Schreiben eröffnet.
- ▶ Aliase werden nun auch innerhalb von anderen Aliase substituiert, z.B. würde für

```
alias car=auto auto=wagen
```

der Aufruf **car** zum Aufruf **wagen** führen.

Jedoch werden Aliase nicht rekursiv substituiert; somit sind Alias-Definitionen wie **alias ls='ls -CF'** möglich.

- ▶ `~+` und `~-` werden nun nach **\$PWD** und **\$OLDPWD** expandiert. Vorher wurden sie bereits zum Zeitpunkt des Lesens durch das momentane und das vorherige Working-Directory ersetzt. Dies führte zu unerwarteten Ergebnissen, insbesondere innerhalb von Skripts und Funktionen.
- ▶ Für `~` wird keine Tilde-Expandierung mehr durchgeführt, wenn es nach einem `:` (Doppelpunkt) angegeben ist.
- ▶ Werden bei den builtin-Kommandos **alias**, **export**, **readonly** und **typeset** als Argumente Variablen-Zuweisungen angegeben, so wird für diese Zuweisungen Tilde-Expandierung, aber keine Wort-Zerteilung und auch keine Parameter-Substitution durchgeführt.
- ▶ Wenn das letzte Element einer Pipeline ein builtin-Kommando, ein Programmiersprach-Kommando oder eine Funktion ist, dann wird es in der momentan aktiven Shell ausgeführt. Zuvor wurde es in einer eigenen Subshell ausgeführt. Alle anderen Elemente einer Pipeline werden wie zuvor in einer eigenen Subshell ausgeführt. Um die alte Arbeitsweise nachzubilden, müßte die Kommandoklammerung `(..)` verwendet werden.

### 5.29.5 Überflüssige Konstrukte

Aus Kompatibilitätsgründen bietet die **ksh** zwar noch die folgenden Konstrukte an. Sie sollten allerdings nicht mehr benutzt werden, da sie durch neue Konstrukte überflüssig geworden sind und eventuell in neuen **ksh**-Versionen entfernt werden. Nachfolgend wird nach den Konstrukten in Klammern angegeben, welche neuen Konstrukte anstelle der alten verwendet werden sollten:

- ▶ Optionen **-k** und **-t** bei **set** (keine)
- ▶ **echo** (**print**)
- ▶ **test** und **[..]** (**[[..]]**)
- ▶ Kommandosubstitution ``kdoliste`` (**\$(kdoliste)**)
- ▶ `funktionsname()` (**(function funktionsname)**)

## 5.30 Literaturhinweis

Als Vorlage zu diesem Kapitel diene das Buch der beiden **ksh**-Entwickler Morris I. Bolsky und David G. Korn:

*The KornShell, Command and Programming Language,*  
Morris I. Bolsky und David G. Korn,  
Prentice Hall, Englewood Cliffs, New Jersey, 1989

# 6 Die C-Shell

*Quid prosunt multa cubicula ?  
In uno iacetis.  
(Wozu die vielen Schlafzimmer ?  
Ihr könnt ja doch nur in einem schlafen.)*

*Seneca*

Die C-Shell wurde auf der starken BSD-UNIX Linie<sup>1</sup> entwickelt. Im Gegensatz zur Korn-Shell ist die C-Shell nicht aufwärtskompatibel zur Bourne-Shell. Die C-Shell, die heute auf UNIX System V schon standardmäßig als Alternative zur Bourne-Shell mitgeliefert wird, verfügt über viele nützliche und hilfreiche Mechanismen, die in der Bourne-Shell doch schmerzlich vermißt werden:

► **History-Mechanismus**

► **Alias-Mechanismus**

► **Job-Kontrolle**

In der C-Shell ist es möglich, Jobs (Programme) zu stoppen und ihre Ausführung vom Vorder- in den Hintergrund bzw. umgekehrt zu verlegen.

► **Tilde-Expandierung**

Auf das Home-Directory eines beliebigen Benutzers kann durch die Angabe `~loginname` zugegriffen werden, ohne daß dazu der Pfadname dieses Directories bekannt sein muß.

► **Eingebaute Arithmetik für ganze Zahlen**

► **Arrays**

► **Automatisches Erkennen von Bourne-Shellskripts**

Da Bourne-Shellskripts nahezu auf allen UNIX-Systemen vorhanden sind, wurde die C-Shell so konzipiert, daß sie Bourne-Skripts als solche erkennen kann, um sie dann von der Bourne-Shell ausführen zu lassen. Damit die C-Shell Bourne-Skripts erkennen kann, wurde folgende Konvention getroffen: In allen **csh**-Skripts muß als erstes Zeichen das Kommentar-Zeichen `#` angegeben sein. Ist dies nicht der Fall, so nimmt die C-Shell an, daß es sich um ein Bourne-Skript handelt und läßt es von der Bourne-Shell ausführen.

---

1. BSD = **B**erkeley **S**ystem **D**istribution.



Trotz dieser Vorzüge ist damit zu rechnen, daß die C-Shell in den nächsten Jahren durch das immer stärkere Aufkommen der Korn-Shell etwas in den Hintergrund gedrängt wird, da sie nicht den Vorteil der Aufwärts-Kompatibilität zur Bourne-Shell bietet.

Die Beschreibung der C-Shell konzentriert sich hier im wesentlichen auf die Syntax, da die grundlegenden Shell-Konzepte bereits ausführlich in den vorherigen Kapiteln behandelt wurden.

## 6.1 Starten und Beenden der C-Shell

Welche Shell als Login-Shell nach dem Anmelden zu starten ist, kann der Systemadministrator durch einen Eintrag in der entsprechenden Benutzerzeile in der Datei */etc/passwd* festlegen<sup>1</sup>. Ist die C-Shell nicht die Login-Shell, so kann sie mit ihrem Programmnamen

**cs****h**

aufgerufen werden. Erhält man in diesem Fall die Meldung

*cs***h**: *not found*

muß nach ihr gesucht werden. Üblicherweise befindet sich das **cs****h**-Programm in */bin*. Andere Unterbringungsmöglichkeiten könnten */usr/bin*, */usr/lbin*, */usr/local* oder */usr/add-on* sein. Jedenfalls sollte das entsprechende Directory in der Variablen **path**<sup>2</sup> mit aufgenommen werden, damit die C-Shell von nun ab nur mit ihrem Programmnamen **cs****h** und nicht nur mit ihrem absoluten Pfadnamen aufgerufen werden kann.

Wird die C-Shell als Login-Shell gestartet, so führt sie zunächst die beiden Dateien *.cshrc* und *.login* aus, wenn diese im Home-Directory vorhanden sind. Wird **cs****h** nicht als Login-Shell, sondern explizit aufgerufen oder über einen Skript-Aufruf gestartet, so werden nur die Kommandos aus *.cshrc* (im Home-Directory) ausgeführt.

Danach meldet sich die **cs****h** mit ihrem Primär-Promptstring "% " <sup>3</sup>, um anzuzeigen, daß sie für die Entgegennahme von Kommandos bereit ist.

Mit dem Befehl

**set prompt** = "text"

kann ein eigener Primär-Promptstring festgelegt werden.

---

1. Siehe Kapitel 2.

2. Vordefinierte Variablen der **cs****h** werden im Unterschied zur Bourne-Shell klein geschrieben.

3. Im Falle des Superusers ist der Primär-Promptstring auf "# " voreingestellt.

Die **csh** kann auf drei verschiedene Arten verlassen werden:

*EOF* (Strg-D)

**exit**

**logout**

**logout** beendet die **csh** jedoch nur, wenn es eine Login-Shell ist.

Um ein versehentliches Beenden der **csh** durch *EOF* zu unterbinden, muß die **csh**-Variable **ignoreeof** gesetzt werden. Ist diese Variable gesetzt, dann kann die **csh** nur noch mit den Kommandos **exit** oder **logout** verlassen werden.

Wird eine **csh** beendet, die eine Login-Shell ist, werden vor der Beendigung von **csh** noch die Kommandos aus der Datei *.logout* ausgeführt. So ist es möglich, noch anfallende Aufräumarbeiten vor dem Ende der Login-Sitzung automatisch durchführen zu lassen. Andere Anwendungsfälle sind z.B. Ende-Zeitpunkt der UNIX-Sitzung festhalten oder Termin-Erinnerungen für die nächste UNIX-Sitzung eintragen lassen.

Für die nachfolgenden Beispiele wird das Directory */user1/egon/cshueb* benutzt:

```
% mkdir cshueb
% cd cshueb
% pwd
/user1/egon/cshueb
%
```

## 6.2 Metazeichen

In der C-Shell existieren weitgehend dieselben Metazeichen wie in der Bourne-Shell. Allerdings sind auch neue Metazeichen hinzugekommen.

Die nachfolgenden Tabellen enthalten nicht nur eine vollständige Übersicht dieser Metazeichen, sondern geben auch bereits eine Kurzbeschreibung ihrer Sonderbedeutung.

Die folgenden Metazeichen entsprechen weitgehend denen der Bourne-Shell:

| Metazeichen | Bedeutung                                                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| >datei      | Standardausgabe in <i>datei</i> umlenken.                                                                                                               |
| >>>datei    | Standardausgabe am Ende von <i>datei</i> anfügen.                                                                                                       |
| <datei      | Standardeingabe auf <i>datei</i> umlenken.                                                                                                              |
| <<wort      | Hier-Dokument (engl.: <i>here document</i> ): Standardeingabe besteht aus den nächsten Eingabezeilen bis zur ersten Zeile, die nur <i>wort</i> enthält. |

Tabelle 6.1: Zur Bourne-Shell identische Metazeichen der C-Shell

| Metazeichen                 | Bedeutung                                                                                                                                                     |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>kdo1</i>   <i>kdo2</i>   | Standardausgabe von <i>kdo1</i> über eine Pipe in die Standardeingabe von <i>kdo2</i> weiterleiten.                                                           |
| *                           | steht für »kein, ein oder mehrere Zeichen«.                                                                                                                   |
| ?                           | steht für »ein beliebiges Zeichen«.                                                                                                                           |
| [...]                       | steht für »eines der in [...] angegebenen Zeichen«; Bereichsangaben wie 0-9 oder a-z sind innerhalb von [...] erlaubt; in <b>csh</b> ist [...] nicht erlaubt. |
| <i>kdo1</i> ; <i>kdo2</i>   | Semikolon trennt mehrere Kommandos in einer Kommandozeile; nach Beendigung von <i>kdo1</i> wird <i>kdo2</i> ausgeführt.                                       |
| <i>kdo</i> &                | <i>kdo</i> im Hintergrund (parallel) ablaufen lassen; in diesem Fall wartet die Shell nicht auf die Beendigung von <i>kdo</i> .                               |
| ` <i>kdo</i> `              | Kommandosubstitution: führt <i>kdo</i> aus und ersetzt dann in der Kommandozeile ` <i>kdo</i> ` durch die Standardausgabe von <i>kdo</i> .                    |
| ( <i>kdo</i> )              | <i>kdo</i> in einer Subshell ausführen; in <b>csh</b> ist { <i>kdo</i> ; } nicht erlaubt.                                                                     |
| <b>\$0, \$1,..., \$9</b>    | Werte der Positionsparameter für ein Shell-Skript.                                                                                                            |
| <b>\$var</b>                | Wert der Shell-Variablen <i>var</i> .                                                                                                                         |
| <b>\${var}text</b>          | Wert der Shell-Variablen <i>var</i> mit nachfolgendem <i>text</i> zu einer Zeichenkette zusammenfügen.                                                        |
| \c                          | Sonderbedeutung des nachfolgenden Metazeichens <i>c</i> ausschalten.                                                                                          |
| '...'                       | Sonderbedeutung aller in '...' angegebenen Zeichen (außer Apostroph ') wird ausgeschaltet.                                                                    |
| "..."                       | Sonderbedeutung aller in "..." angegebenen Zeichen, außer \$ und `...` wird ausgeschaltet.                                                                    |
| #                           | leitet Kommentar ein; Rest der Zeile wird von der <b>csh</b> ignoriert; Kommentare sind in der <b>csh</b> nur in Skripten erlaubt.                            |
| <i>kdo1</i> && <i>kdo2</i>  | <i>kdo2</i> wird nur dann ausgeführt, wenn Ausführung von <i>kdo1</i> erfolgreich verlief.                                                                    |
| <i>kdo1</i>     <i>kdo2</i> | <i>kdo2</i> wird nur dann ausgeführt, wenn Ausführung von <i>kdo1</i> nicht erfolgreich verlief.                                                              |
| Neuzeile-Zeichen            | übergibt eine Kommandozeile an die Shell zur Abarbeitung.                                                                                                     |
| Leerzeichen                 |                                                                                                                                                               |
| Tabulatorzeichen            | Trennzeichen für Wörter (sonst keine Sonderbedeutung).                                                                                                        |

Tabelle 6.1: Zur Bourne-Shell identische Metazeichen der C-Shell (Fortsetzung)

Neu gegenüber der Bourne-Shell sind die folgenden Metazeichen:

| Metazeichen                         | Bedeutung                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&gt;! datei</code>            | wie bei <code>&gt;</code> wird die Standardausgabe in <i>datei</i> umgelenkt. Im Unterschied zu <code>&gt;</code> wird jedoch der alte Inhalt von <i>datei</i> in jedem Fall überschrieben, selbst wenn die Variable <b>noclobber</b> gesetzt ist.                                                                                                                         |
| <code>&gt;&amp; datei</code>        | lenkt sowohl die Standardausgabe als auch die Standardfehlerausgabe in die angegebene <i>datei</i> um.                                                                                                                                                                                                                                                                     |
| <code>&gt;&amp;! datei</code>       | wie <code>&gt;&amp;</code> , nur daß eine bereits existierende <i>datei</i> auch dann überschrieben wird, wenn <b>noclobber</b> gesetzt ist.                                                                                                                                                                                                                               |
| <code>&gt;&gt;! datei</code>        | wie bei <code>&gt;&gt;</code> wird die Standardausgabe ans Ende von <i>datei</i> umgelenkt. Während jedoch bei <code>&gt;&gt;</code> , wenn die entsprechende <i>datei</i> nicht existiert und <b>noclobber</b> gesetzt ist, eine Fehlermeldung ausgegeben wird, wird bei <code>&gt;&gt;!</code> in jedem Fall (ohne Fehlermeldung) in die Datei <i>datei</i> geschrieben. |
| <code>&gt;&gt;&amp; datei</code>    | lenkt sowohl die Standardausgabe als auch die Standardfehlerausgabe an das Ende von Datei <i>datei</i> um.                                                                                                                                                                                                                                                                 |
| <code>&gt;&gt;&amp;! datei</code>   | wie <code>&gt;&gt;&amp;</code> , nur daß auch dann in <i>datei</i> geschrieben wird, wenn <i>datei</i> noch nicht existiert und <b>noclobber</b> gesetzt ist.                                                                                                                                                                                                              |
| <code>\$(10), \${11}, ..</code>     | Zugriff auf die Werte der Positionsparameter 10, 11, ...                                                                                                                                                                                                                                                                                                                   |
| <code>{pattern1,pattern2,..}</code> | deckt jeden String ab, der durch eines der angegebenen Pattern <i>pattern1</i> , <i>pattern2</i> , .. abgedeckt wird.                                                                                                                                                                                                                                                      |
| <code>~</code>                      | Tilde-Expandierung: <code>~loginname</code> wird durch das Home-Directory des Benutzers <i>loginname</i> ersetzt und <code>~</code> bzw. <code>~/..</code> wird zu <b>\$home</b> bzw. <b>\$home/..</b> expandiert.                                                                                                                                                         |
| <code>kdo1   &amp; kdo2</code>      | Standardausgabe und Standardfehlerausgabe von <i>kdo1</i> werden über eine Pipe in die Standardeingabe von <i>kdo2</i> geleitet.                                                                                                                                                                                                                                           |
| <code>(ausdruck)</code>             | Klammerung eines Ausdrucks ist bei den builtin-Kommandos <b>exit</b> , <b>if</b> und <b>while</b> notwendig.                                                                                                                                                                                                                                                               |
| <code>!</code>                      | leitet History-Substitutionen ein.                                                                                                                                                                                                                                                                                                                                         |
| <code>^</code>                      | Metazeichen für History-Substitution.                                                                                                                                                                                                                                                                                                                                      |
| <code>:</code>                      | muß vor Modifikatoren angegeben werden.                                                                                                                                                                                                                                                                                                                                    |

Tabelle 6.2: Neue Metazeichen der C-Shell

## 6.3 Einfache Kommandos, Pipelines und Listen

*Listen* werden aus einer oder mehreren *Pipelines* gebildet.

*Pipelines* wiederum setzen sich aus einem oder mehreren einfachen Kommandos zusammen.

Ein *einfaches Kommando* ist eine Folge von Wörtern, wobei das erste Wort den Namen des Kommandos angibt, das auszuführen ist.

### 6.3.1 Einfache Kommandos und exit-Werte

Ein *einfaches Kommando* ist eine Folge von Wörtern, die durch Leer- und/oder Tabulatorzeichen voneinander getrennt sind. Das erste Wort gibt den Namen des auszuführenden Kommandos an.

Der Rückgabewert eines Kommandos wird als sein *exit-Status* bezeichnet. Der *exit-Status* eines Kommandos zeigt immer den Erfolgsgrad der Kommandoausführung an:

| Rückgabewert      | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0                 | zeigt an, daß entsprechendes Kommando erfolgreich ausgeführt wurde.                                                                                                                                                                                                                                                                                                                                                                                                 |
| verschieden von 0 | zeigt an, daß entsprechendes Kommando nicht erfolgreich ausgeführt werden konnte.<br><br>Der <i>exit-Status</i> liefert in diesem Fall noch zusätzlich die Information, ob das Kommando <u>normal</u> (z.B. Datei nicht vorhanden) oder <u>abnormal</u> (z.B. bei Division durch 0 oder bei Abbruch durch den Benutzer) beendet wurde. Wurde ein Programm abnormal beendet, so wird auf den eigentlichen <i>exit-Status</i> noch 0200 (oktal) <sup>a</sup> addiert. |

a. Dezimal: 128.

Mit **echo \$status** kann der *exit-Status* des zuletzt ausgeführten Kommandos ausgegeben werden.

**Beispiel**

```
% cd /abc
/abc: No such file or directory
% echo $status
1
% cd
% echo $status
0
% cd cshueb
% echo $status
0
%
```

### 6.3.2 Pipelines

Eine Pipeline ist eine Folge von einem oder mehreren Kommandos, welche mit `|` oder `&` voneinander getrennt sind. Das Pipesymbol `|` bewirkt, daß die Standardausgabe des links vom Pipesymbol `|` angegebenen Kommandos direkt in die Standardeingabe des rechts davon stehenden Kommandos weitergeleitet wird. Wird das Pipesymbol `&` angegeben, so wird nicht nur die Standardausgabe, sondern auch die Standardfehlerausgabe in die Standardeingabe des rechts davon stehenden Kommandos weitergeleitet.

Der exit-Status einer Pipeline ist der exit-Status des zuletzt in der Pipeline angegebenen Kommandos.

### 6.3.3 Listen

Eine *Liste* ist eine Folge von ein oder mehreren Pipelines, welche durch die Zeichen

`;`, `&`, `&&` oder `||`

voneinander getrennt sind. `&` und `;` dürfen dabei auch am Ende einer solchen Liste angegeben sein.

Diese Trennzeichen haben dabei folgende Bedeutung:

| Trennzeichen            | Bedeutung                                                                                                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>;</code>          | bewirkt, daß die in der Liste angegebenen Pipelines streng nacheinander ausgeführt werden.                                                                                           |
| <code>&amp;</code>      | bewirkt, daß die davor angegebene Pipeline <sup>a</sup> im Hintergrund (asynchron) ausgeführt wird.                                                                                  |
| <code>&amp;&amp;</code> | bewirkt, daß die danach angegebenen Pipelines nur dann ausgeführt werden, wenn die vorherigen Pipelines erfolgreich (exit-Status = 0) ausgeführt werden konnten.                     |
| <code>  </code>         | bewirkt, daß die danach angegebenen Pipelines nur dann ausgeführt werden, wenn die vorherigen Pipelines nicht erfolgreich (exit-Status verschieden von 0) ausgeführt werden konnten. |

a. Eine Pipeline kann auch aus nur einem Kommando bestehen.

Die Vorrangregeln der hier angegebenen Trennzeichen von Pipelines in einer Liste sind: `(; gleich &)` < `(&& gleich ||)`

**Hinweis**

Um eine Liste als einfaches Kommando behandeln zu können, muß diese mit **(..)** geklammert werden:

```
% ls /bin/e*; ls /bin/f* | wc -l
/bin/echo
/bin/ed
/bin/env
/bin/expr
4
% (ls /bin/e*; ls /bin/f*) | wc -l
8
%
```

## 6.4 Kommentare

Wie in der Bourne-Shell gilt: Wenn ein Wort mit **#** beginnt, dann werden dieses Wort und alle nachfolgenden Zeichen dieser Zeile von der **csh** als Kommentar interpretiert und als solcher von ihr ignoriert.

Anders als in der Bourne-Shell ist die Angabe von Kommentaren in der Kommandozeile einer interaktiven **csh** nicht erlaubt.

Zudem hat das Metazeichen **#** in der **csh** eine weitere Sonderbedeutung, wenn es an erster Stelle eines Shell-Skripts angegeben ist. In diesem Fall wird das betreffende Shell-Skript von der **csh** ausgeführt. Ist als erstes Zeichen in einem Skript nicht **#** angegeben, so nimmt die **csh** an, daß es sich um ein Bourne-Shellskript handelt, und läßt es von der Bourne-Shell ausführen:

```
% pwd
/user1/egon/cshueb
% set path = (/bin /usr/bin .) [path-Variable für csh setzen]
% cat typ
type $1 # sh-builtin: Typ von $1 bestimmen
% chmod u+x typ
% type shift
type: Command not found. [type ist kein der csh bekanntes Kommando]
% typ shift [typ wird von Bourne-Shell ausgeführt]
shift is a shell builtin [type ist sh-Kommando]
% cat typ2
csh-Skript
type $1 # sh-builtin: Typ von $1 bestimmen
% chmod u+x typ2
% typ2 shift [typ2 wird von csh ausgeführt]
type: Command not found. [type ist kein der csh bekanntes Kommando]
%
```

## 6.5 Shell-Skripts (Shell-Prozeduren)

Kommandos können nicht nur interaktiv eingegeben, sondern auch in eine Datei geschrieben und diese Datei dann der **cs**h zur Abarbeitung der darin angegebenen Kommandos vorgelegt werden.

Solche Kommandodateien werden auch unter der C-Shell als *Shell-Skripts* (oder *Shell-Prozeduren*) bezeichnet.

Es existieren mehrere Möglichkeiten, solche Shell-Skripts zu starten:

1. **cs**h *skript-name*

Wird ein Shell-Skript mit **cs**h aufgerufen, so wird eine neue Subshell<sup>1</sup> gestartet, die das angegebene Shell-Skript ausführt.

2. *skript-name*

Bei dieser Aufrufform muß allerdings die Datei, in welcher das Shell-Skript gespeichert ist, ausführbar sein, was beim Aufruf mit **cs**h nicht erforderlich ist. Falls die entsprechende Shell-Skript-Datei nicht ausführbar ist, so muß sie zuerst mit dem Kommando **ch**mod dazu gemacht werden.

Wie beim Aufruf mit **cs**h wird auch hier eine eigene Subshell gestartet, welche die Ausführung des aufgerufenen Shell-Skripts übernimmt.

**Hinweise**

Als Dateinamen für **cs**h-Skripts sollten keine Namen von Unix-Kommandos gewählt werden.

Neben den beiden Aufrufformen, die zuvor angegeben wurden, könnte ein **cs**h-Skript auch mit

**cs**h <*skript-name*

zur Ausführung gebracht werden. In diesem Fall ist für die Dauer der Skript-Ausführung die Standardeingabe nicht mehr auf das Terminal eingestellt, sondern in die Skript-Datei umgelenkt.

## 6.6 Kommandosubstitution

Kommandos, deren Standardausgabe von der **cs**h als Teil der Kommandozeile zu verwenden ist, müssen mit »Gegen-Apostroph« (engl.: *backquotes* oder *accents graves*):

``kommandos``

geklammert werden.

Alle Metazeichen innerhalb einer Kommandosubstitution behalten ihre Sonderbedeutung.

---

1. Kindprozeß zur aktuellen Shell.



**Hinweise**

Normalerweise werden Leer-, Tabulator- und Neuezeile-Zeichen als Wort-Trennzeichen für die Ausgabe der *kommandos* verwendet. Wird jedoch die Kommandosubstitution mit "." geklammert, wird nur das Neuezeile-Zeichen als Wort-Trennzeichen verwendet; Leer- und Tabulatorzeichen gehören dann zu einem Wort und bleiben erhalten:

```
% wc `ls -C`
1 7 33 typ
2 9 46 typ2
3 16 79 total
% wc "`ls -C`"
wc: cannot open typ typ2 ["typ typ2" zählt hierbei als ein Wort]
%
```

Da das letzte Neuezeile-Zeichen bei der Kommandosubstitution immer entfernt wird, ist es möglich, Kommandosubstitution als Teil eines Wortes zu verwenden:

```
% echo `pwd`-Directory
/user1/egon/cshueb-Directory
% echo `pwd`/t*
/user1/egon/cshueb/typ /user1/egon/cshueb/typ2
%
```

## 6.7 Shell-Parameter

Die C-Shell kennt zwei Arten von Parametern:

### Positionsparameter

Ihr Name wird als Ziffer 0, 1, 2, ..., 9 angegeben. Wie später gezeigt wird, ist in der **csh** auch ein Zugriff auf die Positionsparameter **10**, **11** usw. mit der Angabe **\$(10)**, **\$(11)** usw. möglich.

### Shell-Variablen

Ihr Name ist bis auf wenige Ausnahmen ein Bezeichner (siehe Kapitel 3)

Wie bei der Bourne-Shell wird durch Voranstellen von **\$** vor einem Parameternamen dessen Wert angesprochen:

**\$parameter** entspricht: Wert von *parameter*

### 6.7.1 Positionsparameter

Positionsparameter stellen wie in der Bourne-Shell die an ein **csh**-Skript übergebenen Argumente zur Verfügung, wobei das 1. Argument dem Parameter 1, das 2. Argument dem Parameter 2 usw. zugewiesen wird. Dem Parameter 0 wird der Name des aufgerufenen **csh**-Skripts zugewiesen.

Auf die Werte der einzelnen Parameternamen kann wieder durch Voranstellen des **\$**-Zeichens zugegriffen werden.

**Beispiel**

```
% pwd ↵
/user1/egon/cshueb
% cat ausgab ↵
#!/bin/csh
echo Das erste Argument ist $1
echo Das zweite Argument ist $2
echo Das Skriptname ist $0
% chmod u+x ausgab ↵
% ausgab hans fritz franz ↵
Das erste Argument ist hans
Das zweite Argument ist fritz
Das Skriptname ist ausgab
$
```

Allerdings ist in der **csh** der Zugriff auf Positionsparameter nicht nur mit **\$1**, **\$2** usw., sondern auch mit **\$argv[1]**, **\$argv[2]** usw. möglich. Darauf wird näher bei der Vorstellung der automatischen Variablen eingegangen.

Mit dem builtin-Kommando **shift** können die Werte der Positionsparameter um eine Position (nach links) verschoben werden. Das Verschieben um mehrere Positionen wie in der Bourne-Shell ist in der **csh** nicht möglich.

### 6.7.2 Shell-Variablen (Schlüsselwort-Parameter)

Der Name einer C-Shell-Variablen ist bis auf wenige Ausnahmen in Form eines Bezeichners (siehe auch Kapitel 3) anzugeben. Manche **csh**-Versionen begrenzen die Länge eines Variablennamens auf 20, andere wiederum lassen längere Namen zu.

Die Werte von Variablen werden als Strings gespeichert. Es ist jedoch möglich, solche Strings als Zahlen oder als logische Werte interpretieren zu lassen.

**csh**-Variablen können auf drei verschiedene Arten definiert werden:

- ▶ **set** *variablenname* [= *wert*] [*variablenname* [= *wert*]]...<sup>1</sup>  
Der Gültigkeitsbereich der Variablen *variablenname*(*n*) ist dabei auf die lokale **csh** begrenzt.
- ▶ **setenv** *variablenname wert*  
Der Gültigkeitsbereich der Variablen *variablenname* ist dabei global. Das heißt, daß diese Variable an Subshells exportiert wird.

---

1. Vor und hinter dem Gleichheitszeichen dürfen hier Trennzeichen angegeben sein.

- ▶ `@variablenname = ausdruck` [`variablenname = ausdruck`]...

Der numerische Ausdruck *ausdruck* wird ausgewertet und das Ergebnis der Variablen *variablenname* als Wert zugewiesen. Der Gültigkeitsbereich der Variablen *variablenname(n)* ist dabei auf die lokale **cs**h begrenzt.

Nachfolgend wird wieder unterschieden zwischen:

- ▶ vom Benutzer frei wählbaren und
- ▶ von der **cs**h vordefinierten Variablenamen.

### Frei wählbare Variablenamen

Bezüglich der Wahl von benutzerdefinierten Variablenamen gelten die Regeln für Bezeichner.

Anders als in der Bourne-Shell wird beim Zugriff auf eine nicht definierte Shell-Variable nicht die leere Zeichenkette geliefert, sondern eine Fehlermeldung ausgegeben. Um zu überprüfen, ob eine Variable definiert ist oder nicht, steht der Ausdruck

`$?variablenname`

zur Verfügung. Dieser Ausdruck liefert den Wert 1, wenn die Variable *variablenname* momentan definiert ist, und ansonsten den Wert 0.

#### Beispiel

```
% echo $?zeit ↵
0
% echo $zeit ↵
zeit: Undefined variable.
% set zeit = fruehling ↵
% echo $?zeit ↵
1
%
```

Mit dem Aufruf von **set** ohne Argumente können alle in der momentanen Shell-Umgebung lokal definierten Shell-Variablen angezeigt werden:

```
% set ↵
argv ()
cwd /user1/egon/cshueb 1
home /user1/egon
path (/bin /usr/bin .)
prompt %
shell /bin/csh
status 0
term wyse60
zeit fruehling
%
```

---

1. Auf manchen **cs**h-Versionen wird diese Variable nicht angeboten.

Wie in der Bourne-Shell, so gilt auch in der C-Shell:

- ▶ Soll einer Shell-Variablen eine Zeichenkette zugewiesen werden, in der die Bedeutung aller Metazeichen, wie z.B. \$, Leerzeichen usw. auszuschalten ist, dann ist diese Zeichenkette mit '.' zu klammern.
- ▶ Die Definition einer Shell-Variablen kann mit dem builtin-Kommando **unset** wieder aufgehoben werden.

## Arrays

Die C-Shell läßt auch die Definition von eindimensionalen Arrays zu. Jedoch muß die Länge des entsprechenden Arrays bereits bei der Definition festgelegt werden. Dies geschieht dadurch, daß allen einzelnen Elementen bereits bei der Definition Werte zugewiesen werden, wobei auch die leere Zeichenkette zugewiesen werden kann, wenn »Lücken« im Array benötigt werden. Ein Array wird also wie folgt definiert:

```
set arrayname = (wert1 wert2 ... wertn)
```

Nach dieser Definition hat `arrayname[1]` als Wert `wert1`, `arrayname[2]` als Wert `wert2` usw.

Soll auf den Wert eines Arrayelements zugegriffen werden, so muß dies mit

```
$arrayname[index]
```

erfolgen.

Soll zugleich auf mehrere Array-Elemente zugegriffen werden, so ist dies mit einer Bereichsangabe möglich:

```
$arrayname[index1-index2]
```

Soll der Wert eines Array-Elements geändert werden, so kann dies mit

```
set arrayname[index] = wert
```

erreicht werden.

Der Ausdruck

```
 $#arrayname
```

liefert die Anzahl der Elemente des Arrays `arrayname`.

Die Angabe

```
 $arrayname bzw.
```

```
 $arrayname[*]
```

bewirkt, daß hierfür alle Arrayelemente (mit Leerzeichen getrennt) eingesetzt werden.

Ein Zugriff auf ein Element außerhalb des Arrays resultiert in einer Fehlermeldung.

**Beispiel**

```
% echo $?zahl ↵
0
% set zahl = (one two three four five six) ↵
% echo $?zahl ↵
1
% echo $zahl[3] ↵
three
% set zahl[4] = vier ↵
% echo $zahl[2-4] ↵
two three vier
% echo $zahl[7] ↵
Subscript out of range.
% set zahl[7] = seven ↵
set: Subscript out of range.
% echo $#zahl ↵
6
% echo $zahl ↵
one two three vier five six
% echo $zahl[*] ↵
one two three vier five six
% echo $zahl[$#zahl] ↵
six
% set zahlx = "one two three four five" ↵
% echo $#zahlx ↵
1 [Nur ein Array-Element]
% echo $zahlx ↵
one two three four five
% echo $zahlx[1] ↵
one two three four five
% unset zahl zahlx ↵
% echo $?zahl ↵
0
% echo $zahl ↵
zahl: Undefined variable.
%
```

Anders als in der Bourne-Shell findet bei der Zuweisung eines Werts an eine Shell-Variable Dateinamen-Expandierung statt:

```
% ls /bin/a* ↵
/bin/acctcom
/bin/adb
/bin/ar
/bin/as
/bin/asm
% set anfang = /bin/a* ↵
% echo $anfang ↵
/bin/acctcom /bin/adb /bin/ar /bin/as /bin/asm
```

```
% set 
anfang (/bin/acctcom /bin/adb /bin/ar /bin/as /bin/asm)
argv (
cwd /user1/egon/cshueb
home /user1/egon
path (/bin /usr/bin .)
prompt %
shell /bin/csh
status 0
term wyse60
zeit fruehling
%
```

**Vordefinierte Shell-Variablen**

Wie die Bourne-Shell bietet auch die C-Shell eine Reihe von Variablen an, deren Namen von ihr bereits fest vorgegeben sind. Bei diesen vordefinierten Variablen wird wieder unterschieden zwischen

- ▶ vom Benutzer veränderbaren Shell-Variablen
- ▶ Variablen, die ständig von der C-Shell automatisch gesetzt werden (auch automatische Variablen genannt).

**Vordefinierte, aber änderbare Shell-Variablen**

Im Gegensatz zu den vordefinierten Variablen der Bourne- und der Korn-Shell sind die vordefinierten Variablen der C-Shell klein geschrieben:

| Variable         | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>cdpath</b>    | Enthält die Suchdirectories für das builtin-Kommando <b>cd</b> (entspricht der Bourne-Shellvariablen <b>CDPATH</b> ).<br><br><i>Keine Voreinstellung.</i>                                                                                                                                                                                                                                                                                               |
| <b>echo</b>      | Wenn diese Variable gesetzt ist, dann wird jede Kommandozeile vor ihrer Ausführung nochmals angezeigt. Während bei nicht builtin-Kommandos alle Substitutionen und Expandierungen vor dieser Ausgabe durchgeführt werden, werden bei builtin-Kommandos diese Expandierungen und Substitutionen erst nach der Ausgabe vorgenommen.<br><br>Die Option <b>-x</b> bewirkt ein implizites Setzen dieser Option.<br><br><i>Voreinstellung:</i> nicht gesetzt. |
| <b>histchars</b> | Durch die Zuweisung einer zwei Zeichen langen Zeichenkette können die beiden History-Metazeichen <b>!</b> und <b>^</b> durch andere ersetzt werden. Das in diesem String als erstes angegebene Zeichen ersetzt dann <b>!</b> und das zweite Zeichen ersetzt <b>^</b> (siehe auch Kapitel 6.14).<br><br><i>Voreinstellung:</i> <b>histchars="!^"</b>                                                                                                     |

Tabelle 6.3: Vordefinierte, aber änderbare csh-Variablen

| Variable         | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>history</b>   | <p>Der Inhalt dieser Variablen legt fest, wie viele der zuletzt eingegebenen Kommandos im History-Puffer aufzuheben sind, um sie eventuell später wieder zu verwenden.</p> <p><i>Voreinstellung:</i> Das zuletzt eingegebene Kommando wird immer aufgehoben.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>home</b>      | <p>Enthält das Home-Directory des entsprechenden Benutzers. Bei Verwendung des Tilde-Mechanismus wird für ~ der Wert von <b>home</b> (<b>\$home</b>) eingesetzt.</p> <p><i>Voreinstellung:</i> Wird mit dem Wert der Variablen <b>HOME</b> aus der Standardumgebung besetzt. <b>HOME</b> seinerseits wird beim Anmelden auf einen vom Systemadministrator (in <i>/etc/passwd</i>) festgelegten Pfadnamen gesetzt.</p>                                                                                                                                                                                                                                                                                                                       |
| <b>ignoreeof</b> | <p>Wenn diese Variable gesetzt ist, so kann eine interaktive <b>csH</b> nur mit <b>exit</b> oder <b>logout</b>, aber nicht mit <i>EOF</i> (<i>Strg-D</i>) verlassen werden; wird meist gesetzt, um ein versehentliches Beenden einer interaktiven <b>csH</b> mit <i>Strg-D</i> zu verhindern.</p> <p><i>Voreinstellung:</i> nicht gesetzt.</p>                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>mail</b>      | <p>Enthält die Pfadnamen von mailbox-Dateien. Im Abstand eines bestimmten Zeitintervalls überprüft die <b>csH</b>, ob neue mail in einer der angegebenen mailbox-Dateien angekommen ist. Wenn ja, meldet sie dies mit "You have new mail." oder, wenn mehrere mailbox-Dateien existieren, mit "New mail in dateiname".</p> <p>Wenn das erste Wort des <b>mail</b>-Werts (<b>\$mail[1]</b>) eine Zahl ist, so legt diese das Zeitintervall für die mail-Überprüfung in Sekunden fest.</p> <p><i>Voreinstellung für das Zeitintervall ist:</i> 600 (10 Minuten).</p> <p><i>Voreinstellung für mail:</i> Falls <b>mail</b> nicht gesetzt ist, legt die Variable <b>MAIL</b> aus der Standardumgebung den Pfadnamen der mailbox-Datei fest.</p> |
| <b>noclobber</b> | <p>Wenn gesetzt, ist ein versehentliches Überschreiben einer existierenden Datei mit einfacher Ausgabeumlenkung &gt; bzw. &gt;&amp; nicht möglich. Um ein Überschreiben in diesem Fall zu erzwingen, müßte &gt;! bzw. &gt;&amp;! verwendet werden.</p> <p>Zudem gilt, wenn <b>noclobber</b> gesetzt ist, daß die Verwendung von &gt;&gt; bzw. &gt;&gt;&amp; auf eine nicht existierende Datei nicht möglich ist. In diesem Fall müßte &gt;&gt;! bzw. &gt;&gt;&amp;! verwendet werden.</p> <p><i>Voreinstellung:</i> nicht gesetzt.</p>                                                                                                                                                                                                      |
| <b>noglob</b>    | <p>Wenn gesetzt, so wird die Dateinamen-Expandierung ausgeschaltet: Metazeichen wie * ? [..] {..} werden als einfache Zeichen ohne Sonderbedeutung behandelt.</p> <p><i>Voreinstellung:</i> nicht gesetzt.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

Tabelle 6.3: Vordefinierte, aber änderbare csh-Variablen (Fortsetzung)

| Variable                     | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>nomatch</b>               | <p>Wenn nicht gesetzt, so wertet die <b>cs</b>h das Vorkommen eines Metazeichens für Dateinamen-Expandierung (* ? [..] {..}), das keinen Dateinamen abdeckt, als Fehler und meldet dies mit "No match". Durch das Setzen dieser Variable kann dies verhindert werden.</p> <p>Voreinstellung: nicht gesetzt.</p>                                                                                                                                                                         |
| <b>notify</b>                | <p>Wenn gesetzt, so wird die Beendigung von anderen gleichzeitig ablaufenden Jobs (Hintergrund-Jobs) sofort gemeldet und nicht erst bei der nächsten Prompt-Ausgabe.</p> <p>Voreinstellung: nicht gesetzt.</p>                                                                                                                                                                                                                                                                          |
| <b>path</b>                  | <p>Enthält die Suchdirectories für Programme.</p> <p>Voreinstellung: Wird mit den Werten der Variablen <b>PATH</b> aus der Standardumgebung besetzt.</p>                                                                                                                                                                                                                                                                                                                                |
| <b>prompt</b>                | <p>Legt den Primär-Promptstring für die <b>cs</b>h fest. Kommt im Promptstring ein ! vor, so wird dieses Ausrufezeichen bei der Ausgabe des Promptstrings durch die Kommandonummer (<i>event number</i>) ersetzt. Diese Kommandonummer wird mit jeder vollständig eingegebenen Kommandozeile um 1 weitergezählt. Die Sonderbedeutung von ! im Promptstring kann mit Quoting \! ausgeschaltet werden.</p> <p>Voreinstellung: "% " (beim Superuser: "# ")</p>                             |
| <b>savehist</b> <sup>a</sup> | <p>Legt die Anzahl der Kommandos fest, welche die <b>cs</b>h bei ihrer Beendigung in der Datei <i>.history</i> (im Home-Directory) aufheben soll. Beim nächsten Starten der <b>cs</b>h kann dann der History-Puffer aus dieser Datei sofort geladen werden und somit eine History-Liste über eine Sitzung hinweg gerettet werden.</p> <p>Keine Voreinstellung.</p>                                                                                                                      |
| <b>shell</b>                 | <p>Enthält den Pfadnamen der Shell.</p> <p>Voreinstellung: meist <b>/bin/csh</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>term</b>                  | <p>Enthält die Terminal-Bezeichnung.</p> <p>Voreinstellung: Wird mit dem Wert der Variablen <b>TERM</b> aus der Standardumgebung besetzt.</p>                                                                                                                                                                                                                                                                                                                                           |
| <b>time</b>                  | <p>Wenn gesetzt, wird eine automatische Zeitmessung eingeschaltet. Die in <b>time</b> angegebene Zahl legt dabei eine Zeitmarke in CPU-Sekunden fest. Wird bei der Ausführung eines Kommandos diese Zeitmarke überschritten, werden nach Beendigung des Kommandos drei Zeitwerte:</p> <p><i>user sys elapsed</i></p> <p>und zusätzlich noch eine Prozentzahl ausgegeben, die das Verhältnis von (<i>user + sys</i>) : <i>elapsed</i> anzeigt.</p> <p>Voreinstellung: nicht gesetzt.</p> |

Tabelle 6.3: Vordefinierte, aber änderbare *csh*-Variablen (Fortsetzung)



| Variable       | Bedeutung                                                                                                                                                                                                                  |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>verbose</b> | Wenn gesetzt, wird nach jeder History-Substitution das substituierte Kommando angezeigt, bevor es ausgeführt wird. Mit der Option <b>-v</b> wird diese Variable automatisch gesetzt.<br><br>Voreinstellung: nicht gesetzt. |

Tabelle 6.3: Vordefinierte, aber änderbare csh-Variablen (Fortsetzung)

- a. Auf manchen **csh**-Versionen wird diese Variable nicht angeboten.

Die momentanen Werte aller Shell-Variablen können mit dem Kommando **set** (ohne Angabe von Argumenten) am Bildschirm ausgegeben werden; dabei werden jedoch »boole'sche Variablen« (wie **ignoreeof**, **echo** usw.) nur angezeigt, wenn sie gesetzt sind.

### Automatische Variablen

Die folgenden vordefinierten Variablen werden ständig neu von der **csh** gesetzt. Auf die Werte dieser automatischen Parameter kann wieder durch Voranstellen von **\$** zugegriffen werden:

| Variablenname | Bedeutung                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------|
| <b>\$</b>     | Prozeßnummer (PID) der aktuellen Shell.                                                          |
| <b>*</b>      | Alle Positionsparameter als ein String:<br>" <b>\$*</b> " entspricht " <b>\$1 \$2 \$3 ...</b> ". |

Den nachfolgenden automatischen Variablen kann der Benutzer zwar explizit Werte zuweisen, allerdings macht dies nur selten Sinn. Zum Beispiel enthält die Variable **cwd** immer das momentane Working-Directory. Wenn der Benutzer nun den Wert von **cwd** ändert, bewirkt dies keinen Wechsel des Working-Directorys; zudem wird beim nächsten **cd**-Aufruf **cwd** wieder neu auf das entsprechende Working-Directory gesetzt.

| Variablenname | Bedeutung                                                                                                                                                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>argv</b>   | In diesem Array werden die Positionsparameter gespeichert:<br><b>\$argv[0]</b> (nicht definiert.<br><b>\$0</b> dagegen enthält den Skript-Namen.)<br><b>\$argv[1]</b> entspricht <b>\$1</b><br>:<br><b>\$argv[9]</b> entspricht <b>\$9</b> |

Tabelle 6.4: Automatische csh-Variablen

| Variablenname           | Bedeutung                                                                                                       |
|-------------------------|-----------------------------------------------------------------------------------------------------------------|
|                         | <b>\$argv[10]</b> entspricht <b>\${10}</b>                                                                      |
|                         | <b>\$argv[11]</b> entspricht <b>\${11}</b>                                                                      |
|                         | :                                                                                                               |
|                         | <b>\$argv[*]</b> entspricht <b>\$*</b>                                                                          |
|                         | <b>\$argv</b> entspricht <b>\$*</b>                                                                             |
|                         | <b>\$#argv</b> Anzahl der Positionsparameter                                                                    |
| <b>child</b>            | Die Prozeßnummer (PID) des zuletzt im Hintergrund gestarteten Kommandos (entspricht dem ! in der Bourne-Shell). |
| <b>cwd</b> <sup>a</sup> | Enthält Pfadnamen des momentanen Working-Directorys.                                                            |
| <b>status</b>           | Exit-Status des zuletzt im Vordergrund ausgeführten Kommandos (entspricht dem ? in der Bourne-Shell).           |

Tabelle 6.4: Automatische csh-Variablen (Fortsetzung)

a. Auf manchen **csh**-Versionen wird diese Variable nicht angeboten.

Beispiel

```
% set
argv ()
cwd /user1/egon/cshueb
home /user1/egon
path (/bin /usr/bin /user1/egon/bin /etc .)
prompt %
shell /bin/csh
status 0
term wyse60
% ls /bin/w*
/bin/wc
/bin/who
/bin/write
% set echo
% ls /bin/w*
ls /bin/wc /bin/who /bin/write
/bin/wc
/bin/who
/bin/write
% set noclobber
set noclobber
% cat >> text1
text1: No such file or directory
% cat > ausgab
ausgab: File exists.
% cat >>! text1
cat
Ein bisschen
Text
```

[Nach ! mindestens ein Leerzeichen angeben]

*Strg-D*

```
% echo "Hallo, egon" > text1
text1: File exists.
% echo "Hallo, egon" >! text1 [Nach ! mindestens ein Leerzeichen angeben]
echo Hallo, egon
% cat text1
cat text1
Hallo, egon
% unset echo
unset echo
% cat text1
Hallo, egon
% set prompt = "! >> " [Nach ! ein Leerzeichen angeben]
60>> set time = 5
61>> find / -name "*.c"
.....
.....
2.2u 17.2s 0:32 60%
62>> ls
ausgab
text1
typ
typ2
63>> ls x*
No match.
64>> set nonomatch
65>> ls x*
x* not found
66>> unset nonomatch
67>> set argv = (a b c d e)
68>> echo $?argv
1
69>> echo $#argv
5
70>> echo $*
a b c d e
71>> echo $argv[2-4]
b c d
72>> echo $argv[$#argv]
e
73>> echo $3
c
74>> echo $argv[3]
c
75>> set prompt = "% "
%
```

**Hinweis**

Die Zugriffsarten **\$n** und **\$argv[n]** weisen einen kleinen Unterschied auf. Während ein Zugriff mit **\$argv[n]** auf einen nicht existierenden Positionsparameter in einem Fehler resultiert, wird bei **\$n** nur ein leerer String geliefert.

Die Angabe **\$argv[n-]** resultiert nicht in einem Fehler, selbst wenn weniger als  $n$  Positionsparameter vorhanden sind. Dasselbe gilt für die Angabe **\$argv[m-n]**, wenn weniger als  $m$ , aber mindestens  $n$  Positionsparameter existieren.

**Beispiel**

```
% set argv = (eins zwei drei vier) ↵
% echo $5 ↵
% echo $argv[5] ↵
Subscript out of range.
% echo $argv[2-] ↵
zwei drei vier
% echo $argv[6-7] ↵
Subscript out of range.
% echo $argv[10-4] ↵
%
%
```

**6.7.3 Parametersubstitution**

Die einfachste Zugriffsmöglichkeit auf den Wert einer Variablen ist

**\$variable**

Die **csH** ersetzt diesen Ausdruck dabei durch alle Worte des Wertes von *variable* (das kann auch eine mit **setenv** definierte Environment-Variable sein); die einzelnen Worte werden dabei durch Leerzeichen getrennt.

Daneben bietet die **csH** auch spezielle Zugriffsmöglichkeiten auf die Werte von Variablen:

**Weitere Zugriffsmöglichkeit auf den Wert einer Variablen**

**\${variable}** ist identisch zur Angabe **\$variable**.

Ein Variablenname muß immer dann in **{..}** angegeben werden, wenn

- ▶ dem Variablennamen ein Buchstabe, eine Ziffer oder ein Unterstrich folgt,
- ▶ auf Positionsparameter mit mehr als einer Ziffer (größer als 9) zuzugreifen ist:

```
% set argv = (a b c d e f g h i j k l m n) ↵
% echo ${12} ↵
l
%
%
```

Die nachfolgenden Zugriffsformen dürfen nur für lokale Shell-Variablen, aber nicht für Environment-Variablen, die mit **setenv** definiert wurden, verwendet werden.

## Zugriff auf ein Array-Element

`$variable[index]`

`${variable[index]}`

Der kleinste Index eines Arrays ist immer 1.

Für *index* kann folgendes angegeben werden:

| <i>index</i>       | <i>Bedeutung</i>                                    |
|--------------------|-----------------------------------------------------|
| eine Zahl <i>n</i> | wählt das <i>n</i> .te Array-Element aus.           |
| <i>m-n</i>         | wählt die Array-Elemente <i>m</i> bis <i>n</i> aus. |
| <i>-n</i>          | entspricht <b>1-n</b>                               |
| <i>m-</i>          | entspricht <i>m-<b>\$#variable</b></i>              |
| <b>*</b>           | wählt alle Array-Elemente aus.                      |

**Beispiel**

```
% set woche = (so mo di mi do fr sa)
% echo $woche[2]
mo
% echo $woche[*]
so mo di mi do fr sa
% echo $woche[3-]
di mi do fr sa
% echo $woche[-4]
so mo di mi
%
```

**Hinweis** Die Ausgabe eines leeren Bereichs resultiert nur dann nicht in einem Fehler, wenn das zweite Argument weggelassen wird oder aber im erlaubten Bereich liegt.

```
% echo $woche[7-4]
% echo $woche[19-]
% echo $woche[5-8]
Subscript out of range.
%
```

## Anzahl der Elemente eines Arrays

`$#variable` oder

`${#variable}`

liefert die Anzahl der Elemente des Arrays *variable*.

```
% echo $#woche
7
%
```

**Skriptname**

\$0

**Einzelne Positionsparameter**

$\$n$

$\${n}$

entspricht  $\$argv[n]$  bzw.  $\${argv[n]}$

**Alle Positionsparameter**

$\$*$

entspricht  $\$argv[*]$

**Abfrage, ob Variable gesetzt**

$\${?variable}$

$\$?variable$

liefert 1, wenn *variable* gesetzt ist, und 0, wenn nicht.

**Abfrage, ob Positionsparameter 0 gesetzt**

$\$?0$

liefert 1, wenn  $\$0$  gesetzt ist, und 0, wenn nicht.

**Lesen einer Zeile von der Standardeingabe**

$\$<^1$

Hierfür wird eine von der Standardeingabe gelesene Zeile eingesetzt. Mit dieser Konstruktion ist es also möglich, eine Zeile von der Standardeingabe zu lesen.

**Beispiel**

```
% cat eingabe␣
#!/bin/csh # um sicherzustellen, dass von csh ausgeführt
echo "Gib eine Zeile ein:"
set zeile = $< # wenn $< nicht vorhanden, dann
 # set zeile = `line`
echo "Deine Eingabe ---$zeile---"
echo "Deine Eingabe umfasst $#zeile Woerter"
echo "Das 2.Wort ist ---$zeile[2]---"
% chmod u+x eingabe␣
% eingabe␣
Gib eine Zeile ein:
Eine kleine Zeile␣
Deine Eingabe ---Eine kleine Zeile---
Deine Eingabe umfasst 3 Woerter
Das 2.Wort ist ---kleine---
%
```

---

1. Auf manchen **csh**-Versionen wird dies nicht angeboten.

## Modifikatoren

Die nachfolgenden Modifikatoren können bei einer Parametersubstitution nach einem : (Doppelpunkt) angegeben werden und haben dann die beschriebenen Auswirkungen:

| Modifikator | Auswirkung                                                                                                                                             |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>:h</b>   | ( <b>head</b> ) liefert den Pfadnamen des entsprechenden Worts.                                                                                        |
| <b>:t</b>   | ( <b>tail</b> ) liefert den Basisnamen des entsprechenden Worts (Gegenstück zu <b>:h</b> ).                                                            |
| <b>:r</b>   | ( <b>remove</b> ) schneidet vom entsprechenden Wort eine vorhandene Endung .xxx ab.                                                                    |
| <b>:e</b>   | ( <b>extension</b> ) liefert vom entsprechenden Wort eine eventuell vorhandene Endung .xxx.                                                            |
| <b>:q</b>   | ( <b>quote</b> ) für das entsprechende Wort wird Quoting vorgenommen, so daß keine weiteren Substitutionen oder Dateinamen-Expandierungen stattfinden. |
| <b>:x</b>   | wie <b>:q</b> , nur daß der entsprechende Wert in mehrere Worte aufgeteilt wird. Trennzeichen sind dabei Leer-, Tabulator- und Neuezeile-Zeichen.      |
| <b>:gh</b>  | ( <b>global head</b> ) <b>:h</b> auf alle Werte eines Arrays anwenden.                                                                                 |
| <b>:gt</b>  | ( <b>global tail</b> ) <b>:t</b> auf alle Werte eines Arrays anwenden.                                                                                 |
| <b>:gr</b>  | ( <b>global read</b> ) <b>:r</b> auf alle Werte eines Arrays anwenden.                                                                                 |
| <b>:ge</b>  | ( <b>global extension</b> ) <b>:e</b> auf alle Werte eines Arrays anwenden.                                                                            |

Tabelle 6.5: Modifikatoren in der csh

Wenn die Klammerung {..} verwendet wird, so müssen die Modifikatoren innerhalb der geschweiften Klammern angegeben werden.

Die Modifikatoren dürfen bei den folgenden Konstruktionen nicht angegeben werden:

**`\${variable}`** bzw **`\${variable}`**

**`\${0}`**

**`\${`}**

**`\${<`**

Die Modifikatoren haben bei mit **setenv** definierten Environment-Variablen keine Auswirkung.

```

Beispiel % set a = /usr/bin/vi
% echo $a
/usr/bin/vi
% echo $a:t
vi
% echo $a:h
/usr/bin
% set v = /usr/bin/vi*
% echo $v
/usr/bin/vi /usr/bin/view
% echo $v:h
/usr/bin /usr/bin/view
% echo $v:t
vi /usr/bin/view
% echo $v:gh
/usr/bin /usr/bin
% echo $v:gt
vi view
% set a = "*"
% echo $a
ausgab eingabe text1 typ typ2
% echo $a:q
*
% set a = *
% echo $a:q
ausgab eingabe text1 typ typ2
% echo $a:x
ausgab eingabe text1 typ typ2
% set y = ($a:q)
% echo $y
ausgab eingabe text1 typ typ2
% echo $#y
5
% set y = $a:x
% echo $y
ausgab
% echo $#y
1
%

```

[Modifikator :h wirkt nur auf das 1.Wort]

[Modifikator :t wirkt nur auf das 1.Wort]

[Dateinamen-Exp. findet bereits hier statt]

## 6.8 Expandierung von Dateinamen auf der Kommandozeile

Beim Aufruf eines Kommandos oder Shell-Skripts wird jedes Wort der Kommandozeile von der **cs**h daraufhin untersucht, ob eines der Zeichen \*, ?, [ oder { darin vorkommt oder ob es mit ~ (engl. *tilde*) beginnt. Wird ein solches Wort gefunden, so betrachtet die **cs**h dieses als ein sogenanntes *pattern*, welches eine Vielzahl von Dateinamen abdecken kann.



Jedes in der Kommandozeile gefundene pattern wird dann von der **csh** expandiert, d.h. durch alle Dateinamen ersetzt<sup>1</sup>, die es abdeckt.

Zur Expandierung von Dateinamen stehen die folgenden Metazeichen zur Verfügung:

| Metazeichen                                    | Bedeutung                                                                                                                                                                                                                                                        |
|------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>*</b>                                       | steht für "eine beliebige Zeichenfolge" (auch die leere)                                                                                                                                                                                                         |
| <b>?</b>                                       | steht für "ein beliebiges einzelnes Zeichen"                                                                                                                                                                                                                     |
| <b>[...]</b>                                   | steht für "eines der in [...] angegebenen Zeichen".<br>Bei der Angabe von Zeichen innerhalb von [...] sind auch Bereichsangaben wie [A-Z] oder [0-9] erlaubt.<br>Hinweis: In der <b>csh</b> ist anders als in der Bourne-Shell die Angabe [...] nicht vorhanden. |
| <b>{wort<sub>1</sub>,wort<sub>2</sub>,...}</b> | Jedes der in {...} mit Komma getrennten Worte wird eingesetzt.<br>Z.B. würde <i>prog.{c,o,txt}</i> die Dateien <i>prog.c</i> , <i>prog.o</i> und <i>prog.txt</i> und <i>{add1,subst}.c</i> die Dateien <i>add1.c</i> und <i>subst.c</i> abdecken.                |
| <b>~</b>                                       | (Tilde) Hierfür setzt die <b>csh</b> den Pfadnamen des Home-Directorys ein, wenn ~ als einziges Zeichen eines Wortes vorkommt oder am Anfang eines Wortes steht und ihm / folgt.                                                                                 |
| <b>~loginname</b>                              | (Tilde) Hierfür setzt die <b>csh</b> den Pfadnamen des Home-Directorys des Benutzers <i>loginname</i> ein, wenn diese Konstruktion am Anfang eines Wortes steht.                                                                                                 |

Tabelle 6.6: csh-Metazeichen für Dateinamen-Expandierung

**Hinweise**

Durch Setzen der Variablen **noglob** kann die Dateinamen-Expandierung ausgeschaltet werden.

Für eine Liste von Kommandowörtern (pattern) ist es ein Fehler, wenn kein einziges der angegebenen pattern mindestens einen Dateinamen abdeckt. Es ist jedoch nicht gefordert, daß jedes angegebene pattern Dateinamen abdecken muß; es wäre also kein Fehler, wenn bereits durch ein angegebenes pattern Dateinamen abgedeckt würden, auch wenn die restlichen pattern keine Dateinamen abdecken. Dies gilt jedoch nur für die Metazeichen \*, ? und [. Die Metazeichen ~ und { werden von der **csh** als Kürzel betrachtet.

Ausnahmen zu obigen Regeln:

Einige Zeichenfolgen in Dateinamen werden nur dann abgedeckt, wenn sie explizit im entsprechenden Pattern angegeben wurden:

.(Punkt) am Anfang eines Dateinamens  
/  
/

1. Alphabetisch sortiert.

```
Beispiel % pwd
/user1/egon/cshueb
% echo {ein,aus}gab*
eingabe ausgab
% echo ~
/user1/egon
% echo ~emil
/user1/emil
% echo ~/c*
/user1/egon/cshueb
% ls b* c*
No match.
% ls a* b* c*
ausgab
% ls /bin/{ca,pg,pa}*
/bin/cat
/bin/passwd
% cd ~emil
% pwd
/user1/emil
% cd ~/cshueb
% pwd
/user1/egon/cshueb
% set noglob
% echo ~
~
% cd ~
~: No such file or directory
% pwd
/user1/egon/cshueb
% ls a*
a* not found
% unset noglob
% echo a{x,u}*
ausgab
% echo a{x,y}*
echo: No match.
% echo {}*
ausgab eingabe text1 typ typ2
% echo {ai}*
echo: No match.
% echo ~xx ~egon
Unknown user: xx.
% echo {a,b} {c,d}
a b c d
%
```

## 6.9 Quoting

Mit dem Quoting-Mechanismus kann die Sonderbedeutung von Metazeichen ausgeschaltet werden.

Die **cs**h kennt folgende Quoting-Arten:

1. Voranstellen von \
2. Klammerung mit '.'
3. Klammerung mit '..'

### 6.9.1 Voranstellen von \

Hier gelten die folgenden Regeln:

- ▶ Wird einem der Metazeichen ein \ vorangestellt, verliert dieses Metazeichen seine Sonderbedeutung. Wird ein \ vor einem Neuezeile-Zeichen angegeben, wird dieses Zeichenpaar von der **cs**h entfernt. Wenn sich also eine Kommandozeile über mehr als eine Zeile erstrecken soll, kann \ als Fortsetzungszeichen verwendet werden.
- ▶ Ein \ vor einem Nicht-Metazeichen hat keinerlei Auswirkung, sondern wird lediglich entfernt.
- ▶ In einem Kommentar und innerhalb von '.' bzw. '..' hat \ keine Sonderbedeutung.
- ▶ Innerhalb der Kommandosubstitution (``kdos``) schaltet \ nur die Sonderbedeutung der Metazeichen \$, ` und \ aus.

### 6.9.2 Klammerung mit '.'

Alle Metazeichen zwischen zwei einzelnen Apostrophen<sup>1</sup> (außer einem weiteren Apostroph) verlieren ihre Sonderbedeutung.

### 6.9.3 Klammerung mit '..'

Bei einer Klammerung mit '..' verlieren die meisten, aber nicht alle Metazeichen ihre besondere Bedeutung: innerhalb von '..' behalten nur die Metazeichen

" ` \$

ihre Sonderbedeutung.

---

1. Nicht zu verwechseln mit den Gegen-Apostrophen der Kommandosubstitution `.`

Im Unterschied zur Apostrophen-Klammerung schaltet diese Form der Klammerung also folgendes nicht aus:

- Parametersubstitution ( $\$variable$ : Zugriff auf den Wert von *variable*)
- Kommandosubstitution ( $\`kdos`$ )

**Hinweis**

Die Erkennung von builtin-Kommandos<sup>1</sup> durch die **csh** kann durch Quoting unterbunden werden; für andere Kommandos gilt dies nicht:

```
% \p\w\ld␣ [pwd ist in der csh kein builtin-Kommando]
/user1/egon/cshueb
% "pwd"␣
/user1/egon/cshueb
% 'pwd'␣
/user1/egon/cshueb
% "cd" /bin␣ [cd ist ein builtin-Kommando der csh]
cd: Command not found.
% \p"wd"␣
/user1/egon/cshueb
% 'ls'␣
ausgab
eingabe
text1
typ
typ2
% set a = Hallo␣
% echo "$a"␣
Hallo
% echo "\$a"␣
\Hallo
%
```

Die Erkennung von **alias**-Namen (siehe auch Kapitel 6.15) kann ebenfalls mit Quoting unterbunden werden:

```
% alias ll "ls -CF"␣
% ll␣
ausgab* eingabe* text1 typ* typ2*
% "ll"␣
ll: Command not found.
%
```

---

1. In Kapitel 6.19 sind alle builtin-Kommandos der **csh** zusammengefaßt.

## 6.10 Ein- /Ausgabeumlenkung

Die üblichen Voreinstellungen sind dabei:

|                        |                            |
|------------------------|----------------------------|
| Standardeingabe:       | Dialogstation (Tastatur)   |
| Standardausgabe:       | Dialogstation (Bildschirm) |
| Standardfehlerausgabe: | Dialogstation (Bildschirm) |

Jedem dieser drei Ein- / Ausgabekanäle ist wieder ein Dateideskriptor zugeordnet:

|                                          |   |
|------------------------------------------|---|
| Standardeingabe ( <b>stdin</b> ):        | 0 |
| Standardausgabe ( <b>stdout</b> ):       | 1 |
| Standardfehlerausgabe ( <b>stderr</b> ): | 2 |

Anderen Dateien, die innerhalb eines Programms oder Kommandos explizit eröffnet werden, werden die Dateideskriptoren 3, 4, 5, 6 usw. zugeordnet.

Diese Ein- / Ausgabekanäle können nun auch in der **csh** in Dateien umgelenkt werden:

`<datei`

lenkt die Standardeingabe (von der Dialogstation) in die Datei *datei* um.

`> datei`

`>! datei`

lenkt die Standardausgabe (von der Dialogstation) in die Datei *datei* um.

Existiert *datei* noch nicht, wird sie neu angelegt. Wenn *datei* bereits existiert, so wird sie in jedem Fall überschrieben, wenn die Variable **noclobber** nicht gesetzt ist; ist dagegen **noclobber** gesetzt, so meldet die **csh** bei `> datei` einen Fehler<sup>1</sup>. Bei der Angabe `>! datei` wird allerdings auch, wenn **noclobber** gesetzt ist, der alte Inhalt von *datei* überschrieben.

`>& datei`

`>&! datei`

lenkt sowohl die Standardausgabe als auch die Standardfehlerausgabe in die angegebene *datei* um.

Existiert *datei* noch nicht, wird sie neu angelegt; `>&! datei` muß hierbei verwendet werden, wenn *datei* bereits existiert und in jedem Fall zu überschreiben ist, auch wenn **noclobber** gesetzt ist.

`>> datei`

`>>! datei`

---

1. Außer es handelt sich um eine gerätespezifische Datei.

lenkt die Standardausgabe (von der Dialogstation) in die Datei *datei* um; allerdings wird hierbei der alte Inhalt einer eventuell schon existierenden Datei *datei* nicht überschrieben, sondern die neuen Ausgabedaten werden an das Ende von *datei* geschrieben.

Wenn *datei* noch nicht existiert, meldet die **csH** bei `>> datei` einen Fehler, wenn die Variable **noclobber** gesetzt ist. Bei der Angabe `>>! datei` wird allerdings auch wenn **noclobber** gesetzt ist, eine neue *datei* angelegt.

`>>& datei`

`>>&! datei`

lenkt sowohl die Standardausgabe als auch die Standardfehlerausgabe an das Ende von Datei *datei* um.

Existiert *datei* noch nicht, wird sie neu angelegt; `>>&! datei` muß hierbei verwendet werden, wenn *datei* noch nicht existiert und in jedem Fall anzulegen ist, auch wenn **noclobber** gesetzt ist.

`<<wort`

Hier-Dokument (engl. *here document*): Die Eingabe an die **csH** wird Zeile für Zeile gelesen, bis eine Zeile gefunden wird, welche genau mit *wort* übereinstimmt, oder bis ein **EOF** gelesen wird.

Für *wort* wird keine Parametersubstitution, keine Kommandosubstitution und keine Dateinamen-Expandierung durchgeführt.

Abhängig davon, ob Quoting im *wort* verwendet wird oder nicht, wird die Sonderbedeutung der Metazeichen für die Eingabezeilen ausgeschaltet oder nicht:

### Quoting in *wort*

In den folgenden Eingabezeilen wird die Sonderbedeutung der Metazeichen ausgeschaltet.

#### Beispiel

```
% cat <<"ENDE"
Dies ist ein Hier-Dokument
$home ist das Home-Directory
"ENDE"
Dies ist ein Hier-Dokument
$home ist das Home-Directory
%
```

### Kein Quoting in *wort*

Für die folgenden Eingabezeilen gelten dann folgende Regeln:

1. Parameter- und Kommandosubstitution findet statt.
2. `\` muß verwendet werden, um die Sonderbedeutung der Zeichen `\` sowie `$` und ``` auszuschalten.

#### Beispiel

```
% cat <<ENDE
Dies ist ein Hier-Dokument
$home ist das Home-Directory
ENDE
Dies ist ein Hier-Dokument
/user1/egon ist das Home-Directory
%
```

#### Hinweis

Weitere Angaben nach *wort* sind möglich:

```
% cat <<ENDE | tr '[a-z]' '[A-Z]'
Dies ist ein Hier-Dokument
$home ist das Home-Directory
ENDE
DIES IST EIN HIER-DOKUMENT
/USER1/EGON IST DAS HOME-DIRECTORY
$
```

#### Hinweise

1. Um die Standardausgabe und die Standardfehlerausgabe getrennt in unterschiedliche Dateien umzulenken, muß ein kleiner Trick angewendet werden, der darin besteht, daß die Standardausgabe mit `>` in einer Subshell und die Standardfehlerausgabe mit `>&` in der gerade aktiven Shell umgelenkt wird:

( *kommando* >ausdatei ) >& fehldatei

#### Beispiel

```
% cat lsax
ls a*
ls x*
% chmod u+x lsax
% lsax
ausgab [Standardausgabe]
x* not found [Standardfehlerausgabe]
% (lsax >stdaus) >& stdfeh1
% cat stdaus
ausgab
% cat stdfeh1
x* not found
%
```

2. Im Gegensatz zur Bourne-Shell wird bei im Hintergrund gestarteten Kommandos die Standardeingabe nicht auf `/dev/null` umgelenkt, sondern wird der entsprechende Hintergrund-Prozeß angehalten, wenn er von der Standardeingabe zu lesen versucht. Mit dem Kommando **fg** (siehe auch Kapitel 6.18) kann dieser Prozeß dann in den Vordergrund gebracht werden, um ihn mit der entsprechenden Eingabe zu versorgen.
3. Die Standardfehlerausgabe eines Kommandos kann zusammen mit der Standardausgabe mit `|&` in eine Pipe umgelenkt werden.
4. Vor und nach den Umlenkungsanweisungen können beliebig viele Leer- und Tabulatorzeichen angegeben sein.
5. Die Umlenkungsstrukturen werden bereits vor dem Aufruf des entsprechenden Programms von der **cs**h ausgewertet, so daß das aufgerufene Programm davon keinerlei Notiz nimmt. Nachdem die **cs**h die geforderten Umleitungen vorgenommen hat, werden die Umlenkungsangaben nicht mehr benötigt und deshalb von der **cs**h aus der Kommandozeile entfernt. Somit werden Umlenkungsanweisungen niemals einem Programm oder Skript als Argumente übergeben.
6. Umlenkungsanweisungen können in einem einfachen Kommando an beliebiger Stelle angegeben werden. Üblicherweise werden die Umlenkungen allerdings am Ende eines Kommandos angegeben.
7. Parameter- und Kommandosubstitution werden immer zuerst durchgeführt, bevor die entsprechenden Umlenkungen von der **cs**h vorgenommen werden.
8. Für die bei Ein-/Ausgabeumlenkung angegebene Datei findet nur dann eine Dateinamen-Expandierung statt, wenn diese nur eine Datei liefert, ansonsten meldet die **cs**h einen Fehler.

## 6.11 Ausdrücke

Ausdrücke werden zum einen bei den builtin-Kommandos **if**, **while** und **exit** und zum anderen bei arithmetischen Berechnungen mit dem builtin-Kommando `@` benötigt<sup>1</sup>.

Anders als die Bourne-Shell verfügt die **cs**h über eine eigene Arithmetik, so daß zeitintensive Aufrufe der nicht builtin-Kommandos **expr** bzw. **bc** nicht mehr notwendig sind, um mathematische Berechnungen durchführen zu lassen.

Die **cs**h verwendet am jeweiligen System bei allen Berechnungen die größtmögliche interne Darstellungsform für ganze Zahlen (z.B. 2, 4 oder 8 Bytes). Die **cs**h prüft dabei niemals auf einen eventuellen *Overflow*.

---

1. Siehe Kapitel 6.13 und Kapitel 6.19.



Wie in der Programmiersprache C wird der Wert eines arithmetischen Ausdrucks als TRUE gewertet, wenn er verschieden von 0, und als FALSE, wenn er 0 ist.

### 6.11.1 Operatoren

Für Ausdrücke können folgende von C her bekannten Operatoren verwendet werden:

| Operator        | Bedeutung                                                                        |
|-----------------|----------------------------------------------------------------------------------|
| -               | Minuszeichen                                                                     |
| ~               | Bitweise Negation (Einerkomplement)                                              |
| *               | Multiplikation                                                                   |
| /               | Division                                                                         |
| %               | Modulofunktion                                                                   |
| +               | Addition                                                                         |
| -               | Subtraktion                                                                      |
| <<              | Links-Shift                                                                      |
| >>              | Rechts-Shift                                                                     |
| <               | kleiner                                                                          |
| >               | größer                                                                           |
| <=              | kleiner gleich                                                                   |
| >=              | größer gleich                                                                    |
| ==              | Vergleich von Strings auf Gleichheit                                             |
| !=              | Vergleich von Strings auf Ungleichheit                                           |
| =~              | Vergleich von Strings auf Gleichheit (als rechter Operand ist pattern erlaubt)   |
| !~ <sup>a</sup> | Vergleich von Strings auf Ungleichheit (als rechter Operand ist pattern erlaubt) |
| &               | bitweises AND                                                                    |
| ^               | bitweises XOR                                                                    |
|                 | bitweises OR                                                                     |

Tabelle 6.7: Operatoren für csh-Ausdrücke

a. Nicht auf allen **csh**-Versionen verfügbar.

Die Operatoren `==`, `!=`, `=~` und `!~` führen einen Stringvergleich durch. Die Operatoren `=~` und `!~` entsprechen weitgehend den Operatoren `==` und `!=`, nur daß sie als rechten Operanden auch ein pattern (siehe Kapitel 6.8) zulassen; dabei prüft die **csh** dann, ob dieses pattern den linken Operanden abdeckt (`=~`) oder nicht abdeckt (`!~`).

Zusätzlich können noch die folgenden Ausdrücke angegeben werden:

| Ausdruck               | liefert TRUE, wenn                                     |
|------------------------|--------------------------------------------------------|
| <b>-d</b> <i>datei</i> | <i>datei</i> ein Directory ( <b>d</b> irectory) ist.   |
| <b>-e</b> <i>datei</i> | <i>datei</i> existiert ( <b>e</b> xistence).           |
| <b>-f</b> <i>datei</i> | <i>datei</i> eine normale Datei ( <b>f</b> ile) ist.   |
| <b>-o</b> <i>datei</i> | <i>datei</i> dem Benutzer gehört ( <b>o</b> wnership). |
| <b>-r</b> <i>datei</i> | <i>datei</i> gelesen ( <b>r</b> ead) werden darf.      |
| <b>-w</b> <i>datei</i> | <i>datei</i> beschrieben ( <b>w</b> rite) werden darf. |
| <b>-x</b> <i>datei</i> | <i>datei</i> ausführbar ( <b>e</b> xecute) ist.        |
| <b>-z</b> <i>datei</i> | <i>datei</i> leer ( <b>z</b> ero) ist.                 |

Für die angegebene *datei* wird dabei Kommandosubstitution und Dateinamen-Expandierung durchgeführt.

Alle Ausdrücke können mit den nachfolgenden Operatoren zu neuen Ausdrücken verknüpft werden:

| Ausdruck                                      | Bedeutung                                                                                                                                    |
|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>!</b> <i>ausdr</i>                         | Logischer Negationsoperator                                                                                                                  |
| <i>ausdr1</i> <b>&amp;&amp;</b> <i>ausdr2</i> | AND-Operator                                                                                                                                 |
| <i>ausdr1</i> <b>  </b> <i>ausdr2</i>         | OR-Operator                                                                                                                                  |
| <b>(</b> <i>ausdr</i> <b>)</b>                | Klammerung eines Ausdrucks:<br>Mit Klammerung ist es möglich, eine andere als die durch die Prioritäten vorgegebene Auswertung zu erzwingen. |

Die folgende Tabelle zeigt die voreingestellte **Prioritätsreihenfolge** (höchste zuerst). Alle in einer Zeile angegebenen Operatoren besitzen dabei die gleiche Priorität:

```
()
- (Minuszeichen)
! ~
* / %
+ -
<< >>
<= >= < >
== != =~ !~
&
^
|
&&
||
```

Alle Operatoren mit gleicher Priorität werden von links nach rechts abgearbeitet.

#### Hinweise

Die einzelnen Operanden und Operatoren eines Ausdrucks müssen durch Leerzeichen getrennt werden; dies gilt jedoch nicht für die Operatoren **&**, **|**, **<**, **>**, **(**, **)**.

Bei der Auswertung eines Ausdrucks werden bei manchen **cs**h-Versionen Zahlen, die mit einer **0** beginnen, als Oktalzahlen interpretiert. Die Ergebnisse selbst werden jedoch immer im Dezimalsystem geliefert.

Fehlt ein Operand oder ist als Operand eine leerer String angegeben, setzt die **cs**h hierfür den Wert **0** ein.

Gegenüber der Bourne-Shell hat die **cs**h den Vorteil, daß Metazeichen wie **\***, **(**, **|** usw. in Ausdrücken nicht mit Quoting auszuschalten sind, was sicherlich der besseren Lesbarkeit dient.

Wird der exit-Status eines Kommandos, das auszuführen ist, als Ausdruck benötigt, ist das entsprechende Kommando bzw. die betreffende Kommandoliste mit **{..}** zu klammern. **{ kdoliste }** liefert 1, wenn die Ausführung von **kdoliste** erfolgreich (exit-Status 0) war, ansonsten liefert diese Konstruktion 0:

```
% pwd
/user1/egon/cshueb
% grep hans *
% echo $status
1
% @ var = ({ grep hans * })
% echo $var
0
% grep erste *
ausgab:echo Das erste Argument ist $1
% echo $status
0
```

```
% @ var = ({ grep erste * }) >& /dev/null ↵
% echo $var ↵
1
%
```

## 6.11.2 Das builtin-Kommando @

Mit dem builtin-Kommando @ ist es möglich, arithmetische Auswertungen durchzuführen und das Ergebnis dann Variablen bzw. Array-Elementen zuzuweisen. @ kann auf 3 verschiedene Arten aufgerufen werden:

| Kommando                                          | Werte                                                                                                                                                                                                                                                            |
|---------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| @                                                 | gibt die Werte aller momentan definierten Shellvariablen aus.                                                                                                                                                                                                    |
| @ variable =<br>ausdruck <sup>a</sup>             | setzt die <i>variable</i> mit dem Ergebnis, das die Auswertung von <i>ausdruck</i> liefert.                                                                                                                                                                      |
| @ vari-<br>able[index]<br>= ausdruck <sup>b</sup> | setzt das Array-Element <i>variable[index]</i> mit dem Ergebnis, das die Auswertung von <i>ausdruck</i> liefert. Hierbei ist zu beachten, daß <i>variable[index]</i> bereits existieren muß, also zuvor mit <b>set</b> oder <b>setenv</b> deklariert werden muß. |

- a. Die Leerzeichen vor und nach dem Zuweisungsoperator = sind nicht notwendig.  
b. Die Leerzeichen vor und nach dem Zuweisungsoperator = sind nicht notwendig.

Für = können hierbei auch die von C her bekannten zusammengesetzten Operatoren +=, -=, \*=, /=, %= und ^= verwendet werden. Auch ist die Verwendung der Postfix-Operatoren ++ (Inkrementieren) und -- (Dekrementieren) erlaubt:

@ variable++ bzw. @ variable--

@ variable[index]++ bzw. @ variable[index]--

Im *ausdruck* können die im vorherigen Kapitel vorgestellten Operatoren verwendet werden. Dabei ist lediglich zu beachten, daß bei Verwendung einer der Operatoren <, >, & oder | der *ausdruck* oder zumindest der entsprechende Teilausdruck mit (..) zu klammern ist.

### Beispiel

```
% @ a = 5 ↵
% @ a++ ↵
% echo $a ↵
6
% @ b=$a * 20 ↵
% echo $b ↵
120
% @ x[2] = 10 ↵
x: Undefined variable.
% set x = (a b c d e f g h i j k l m) ↵
% @ x[2] =10 ↵
% echo $x ↵
```

```

a 10 c d e f g h i j k l m
% @ x[2] += 25
% echo $x[2]
35
% @ x[3] += 10
@: Expression syntax.
%
```

Die beiden nachfolgenden **cs**h-Skripts testen die zuvor vorgestellten Operatoren:

```

% cat optest
#!/bin/csh
#
optest - csh-Skript zum Testen der Ausdrucks-Operatoren
#
#-----
#
echo -n "Gib eine Zahl ein: "
set a = $< # evtl.: set a = `line`
echo -n "Gib noch eine Zahl ein: "
set b = $< # evtl.: set b = `line`
@ z = -$a ; echo "-$a = $z"
@ z = (~ $a) ; echo "~ $a = $z"
@ z = $a * $b ; echo "$a * $b = $z"
@ z = $a / $b ; echo "$a / $b = $z"
@ z = $a % $b ; echo "$a % $b = $z"
@ z = $a + $b ; echo "$a + $b = $z"
@ z = $a - $b ; echo "$a - $b = $z"
@ z = ($a << $b) ; echo "$a << $b = $z"
@ z = ($a >> $b) ; echo "$a >> $b = $z"
@ z = ($a < $b) ; echo "$a < $b = $z"
@ z = ($a > $b) ; echo "$a > $b = $z"
@ z = $a == $b ; echo "$a == $b = $z"
@ z = $a != $b ; echo "$a != $b = $z"
@ z = ($a & $b) ; echo "$a & $b = $z"
@ z = ($a ^ $b) ; echo "$a ^ $b = $z"
@ z = ($a | $b) ; echo "$a | $b = $z"
@ z = ($a && $b) ; echo "$a && $b = $z"
@ z = ($a || $b) ; echo "$a || $b = $z"
% chmod u+x optest
% optest
Gib eine Zahl ein: 25
Gib noch eine Zahl ein: 12
-25 = -25
~ 25 = -26
25 * 12 = 300
25 / 12 = 2
25 % 12 = 1
25 + 12 = 37
25 - 12 = 13
```

```

25 << 12 = 102400
25 >> 12 = 0
25 < 12 = 0
25 > 12 = 1
25 == 12 = 0
25 != 12 = 1
25 & 12 = 8
25 ^ 12 = 21
25 | 12 = 29
25 && 12 = 1
25 || 12 = 1
%
```

Das nachfolgende **csh**-Skript *zoptest* macht bereits von der einfachen **if**-Abfrage Gebrauch:

#### **if** (*ausdruck*) *kommando*

Falls die Auswertung von *ausdruck* TRUE ergibt, wird das angegebene *kommando* ausgeführt, sonst nicht. *kommando* muß dabei in der gleichen Zeile wie **if** (*ausdruck*) angegeben sein (siehe auch Kpaitel 6.13.1):

```

% cat zoptest ↵
#!/bin/csh
#
zoptest - csh-Skript zum Testen der String-Operatoren
#
#-----
#
echo -n "Gib einen String ein: "
set a = $< # evtl. angeben: set a = `line`
echo -n "Gib noch einen String ein: "
set b = $< # evtl. angeben: set b = `line`

if ($a == $b) echo "String '$a' ist gleich String '$b'"
if ($a != $b) echo "String '$a' ist nicht gleich String '$b'"
if ($a =~ $b) echo "String '$a' ist gleich String '$b'"
if ($a !~ $b) echo "String '$a' ist nicht gleich String '$b'"
if ($a =~ *[aeiou]*) echo "String '$a' enthaelt einen Vokal"
if ($b !~ *[aeiou]*) echo "String '$b' enthaelt keinen Vokal"
if ($a =~ *$b*) echo "String '$b' ist in '$a' enthalten"
% chmod u+x zoptest ↵
% zoptest ↵
Gib einen String ein: Berlin ↵
Gib noch einen String ein: DHH ↵
String 'Berlin' ist nicht gleich String 'DHH'
String 'Berlin' ist nicht gleich String 'DHH'
String 'Berlin' enthaelt einen Vokal
String 'DHH' enthaelt keinen Vokal
% zoptest ↵
Gib einen String ein: Peterchen ↵
```

```
Gib noch einen String ein: Peter↵
String 'Peterchen' ist nicht gleich String 'Peter'
String 'Peterchen' ist nicht gleich String 'Peter'
String 'Peterchen' enthaelt einen Vokal
String 'Peter' ist in 'Peterchen' enthalten
%
```

Das nachfolgende Shell-Skript *fiba* macht bereits von der später beschriebenen **while**-Schleife (siehe Kapitel 6.13.3) Gebrauch. Es berechnet dabei wieder einmal die Fibonacci-Zahlen (bis 1000):

```
% pwd↵
/user1/egon/cshueb
% cat fiba↵
#!/bin/csh
set x=1 y=1

while ($x <= 1000)
 echo $x
 @ z = $x + $y
 set x=$y y=$z
end
% chmod u+x fiba↵
% fiba↵
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
%
```

## 6.12 Kommandoklammerung mit (..)

Mit (..) geklammerte Kommandos werden in einer Subshell ausgeführt.

**Beispiel**

```
% pwd ↵
/user1/egon/cshueb
% (cd /usr; pwd) ↵
/usr
/user1/egon/cshueb
% pwd ↵
/user1/egon/cshueb
%
```

## 6.13 Kommandos zur Ablaufsteuerung

Die nachfolgend vorgestellten Kommandos werden in der originalen **csh**-Literatur als builtin-Kommandos aufgeführt. Aus Konsistenz zu den anderen Shell-Kapiteln werden sie hier jedoch als Kommandos zur Ablaufsteuerung klassifiziert.

Im Unterschied zur Bourne- und Korn-Shell können die hier vorgestellten Kommandos nicht Element einer Pipeline oder Liste sein, und sie können auch nicht im Hintergrund gestartet werden<sup>1</sup>.

Im Gegensatz zur Bourne-Shell müssen hier die Bedingungen bei **if**, **foreach**, **while** und **switch** wie in C mit (..) geklammert werden.

### 6.13.1 if-Anweisung

Die **if**-Anweisung kann auf drei verschiedene Arten benutzt werden:

(1) **if** (*ausdruck*) *kommando*

(2) **if** (*ausdruck*) **then**  
    *kdos*  
    **endif**

(3) **if** (*ausdruck*) **then**  
    *then\_kdos*  
    **else**  
    *else\_kdos*  
    **endif**

(1) **if** (*ausdruck*) *kommando*

Wenn die Auswertung von *ausdruck* TRUE liefert, wird das angegebene *kommando* ausgeführt, ansonsten wird es nicht ausgeführt.

---

1. Manche **csh**-Versionen haben jedoch diese Schwäche inzwischen beseitigt.



Die vollständige **if**-Anweisung muß in einer Zeile angegeben sein. Soll z.B. *kommando* in einer eigenen Zeile stehen, so muß das Fortsetzungszeichen \ verwendet werden:

```
if (ausdruck) \
 kommando
```

Für *kommando* muß ein einfaches Kommando angegeben sein. Es darf also keine Pipeline, Kommandoliste oder geklammerte Kommandoliste dafür angegeben werden.

Vorsicht: Eventuell für *kommando* angegebene Ein-/Ausgabeumlenkungen werden auch dann von der **csh** ausgewertet, wenn *ausdruck* FALSE liefert und somit *kommando* überhaupt nicht ausgeführt wird.

**Beispiel**

```
% cat wcdat ↵
#!/bin/csh

wcdat - zaehlt alle Dateien in den als
Argumente angegebenen Directorybaeumen.
Sind keine Directories angegeben, so
wird das Working-Directory angenommen.

set tmpdatei = "/tmp/wcdat.$$"

if ($#argv == 0) set argv = (.)

(find $argv -print > $tmpdatei) >& /dev/null
echo "Die Directorybaeume"
echo " $argv"
echo "enthalten `cat $tmpdatei | wc -l` Dateien"
/bin/rm -f $tmpdatei
% chmod u+x wcdat ↵
% wcdat /bin /usr/bin ↵
Die Directorybaeume
 /bin /usr/bin
enthalten 396 Dateien
% wcdat ↵
Die Directorybaeume
 .
enthalten 14 Dateien
%
```

```
(2) if (ausdruck) then
 kdos
 endif
```

Wenn die Auswertung von *ausdruck* TRUE liefert, werden die angegebenen *kdos* ausgeführt, ansonsten werden sie nicht ausgeführt.

```
(3) if (ausdruck) then
 then_kdos
else
 else_kdos
endif
```

Wenn die Auswertung von *ausdruck* TRUE liefert, werden die *then\_kdos* ausgeführt, ansonsten werden die *else\_kdos* ausgeführt.

Das Schlüsselwort **if** muß entweder als erstes Wort einer Zeile oder nach **else** angegeben sein.

Das Schlüsselwort **then** muß dabei in der gleichen Zeile wie **if** (*ausdruck*) stehen.

Die Schlüsselwörter **else** und **endif** müssen als erstes Wort einer Zeile angegeben sein.

Es sind auch beliebige **else-if**-Kaskaden möglich:

```
if (ausdruck) then
 then_kdos1
else if (ausdruck) then
 then_kdos2
else if (ausdruck) then

else
 else_kdos
endif
```

**Beispiel**

Das nachfolgende **csh**-Skript *copy* simuliert das MS-DOS-Kommando **copy**:

```
% cat copy ↵
#!/bin/csh
#
copy - simuliert das MS-DOS Kommando copy
#
Syntax:
copy quell-datei(en) [ziel-datei/directory]
#
if ($#argv == 0) then
 echo "copy : falsche Anzahl von Argumenten"
 echo "usage: copy quell-datei(en) [ziel-datei/directory]"
 exit (1)
else if ($#argv == 1) then
 cp $1 .
else if ($#argv == 2) then
 if (-f $2) then
 echo "$2 wurde durch diesen Aufruf:"
 echo " copy $1 $2 ueberschrieben"
 echo -n "Ist dies beabsichtigt ? (j/n) : "
 if ($< !~ [jj]*) then
```

```
 cp $argv .
 else
 cp $1 $2
 endif
else
 cp $1 $2
endif
else if (-f $argv[${#argv}]) then
 cp $argv .
else
 cp $argv
endif
% chmod u+x copy 
%
```

### 6.13.2 switch-Anweisung

**switch** (*wort*)

**case** *pattern*<sub>1</sub>:

*kdos*<sub>1</sub>

**breaksw**

**case** *pattern*<sub>2</sub>:

*kdos*<sub>2</sub>

**breaksw**

    :

    :

[ **default**:

*kdos*<sub>n</sub>]

**endsw**

Die **csh** führt zuerst für das angegebene *wort* Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung durch. Den daraus resultierenden String vergleicht sie dann nacheinander mit den bei **case** angegebenen *pattern*. Findet sie ein übereinstimmendes *pattern*, führt sie die nachfolgenden *kdos* aus, bis sie **breaksw** liest; **breaksw** bewirkt das Verlassen der **switch**-Anweisung. Wird eine zu einer **case**-Marke angegebene Kommandofolge nicht mit einem **breaksw** beendet, wird die Ausführung – wie in C – mit der Kommandofolge der nächsten **case**-Marke fortgesetzt.

Paßt keines der angegebenen *pattern*, werden, wenn die **default**-Marke vorhanden ist, die danach angegebenen Kommandos ausgeführt, oder eben die **switch**-Anweisung ohne jegliche weitere Aktion verlassen.

In den *pattern* können die Metazeichen \*, ? und [...] der Dateinamen-Expandierung verwendet werden.

Die Schlüsselwörter **switch**, **case**, **endsw** und **default** müssen als erstes Wort einer Zeile angegeben sein.

```
Beispiel % cat loesch ↵
#!/bin/csh
#
loesch - loescht die angegebenen Dateien, macht aber
zuvor eine Sicherheitskopie von diesen im
Directory $home/.muell. Von dort koennten
somit versehentlich geloeschte Dateien
wieder geholt werden.
#
Syntax:
loesch [-l] [-r] [datei(en)]
-l Auflisten aller Dateien aus dem
Muelleimer ($home/.muell)
-r Muelleimer leeren; alle Dateien von
$home/.muell nun wirklich loeschen.
Sind datei(en) angeben, so werden diese
zwar geloecht, aber im Muelleimer
($home/.muell) aufgehoben
#

set muelleimer = $home/.muell

if (! -d $muelleimer) mkdir $muelleimer

if ($#argv == 0) set argv = (-l)

switch ($1)
 case -[lL]*:
 echo "Folgende Dateien sind im Muelleimer:"
 ls -al $muelleimer | pg
 exit (0)
 case -[rR]*:
 echo "Der Muelleimer mit folgenden Dateien:"
 echo $muelleimer/*
 echo "wird geleert."
 /bin/rm -rf $muelleimer/*
 exit (0)
 default:
 mv $argv $muelleimer
 breaksw
endsw

% chmod u+x loesch ↵
% cat hexziff ↵
#!/bin/csh
#
if ($#argv == 0) then
 echo "usage: hexziff ziffer"
 exit(1)
endif
switch ($argv[1])
```

```
case [0-9]:
 echo $1
 breaksw
case [aA]:
 echo 10
 breaksw
case [bB]:
 echo 11
 breaksw
case [cC]:
 echo 12
 breaksw
case [dD]:
 echo 13
 breaksw
case [eE]:
 echo 14
 breaksw
case [fF]:
 echo 15
 breaksw
default:
 echo "Keine erlaubte Hexadezimalziffer"
 breaksw
endsw
% chmod u+x hexziff ↵
% hexziff ↵
usage: hexziff ziffer
% hexziff C ↵
12
% hexziff X ↵
Keine erlaubte Hexadezimalziffer
%
```

Auch bei der **switch**-Anweisung der **cs**h wird das angegebene wort der Reihe nach mit den **case**-pattern verglichen. Deswegen sollte die **default**:-Marke immer als letztes angegeben sein:

```
% cat hexziff2 ↵
#!/bin/csh
#
if ($#argv == 0) then
 echo "usage: hexziff2 ziffer"
 exit(1)
endif

switch ($argv[1])
default:
 echo "Keine erlaubte Hexadezimalziffer"
 breaksw
case [0-9]:
```

```
 echo $1
 breaksw
case [aA]:
 echo 10
 breaksw
case [bB]:
 echo 11
 breaksw
case [cC]:
 echo 12
 breaksw
case [dD]:
 echo 13
 breaksw
case [eE]:
 echo 14
 breaksw
case [fF]:
 echo 15
 breaksw
endsw
% chmod u+x hexziff2
% hexziff2 C
Keine erlaubte Hexadezimalziffer
%
```

### 6.13.3 while-Schleife

**while** (*ausdruck*)

*kdos*

**end**

Solange die Auswertung von *ausdruck* TRUE liefert, werden die *kdos* in der **while**-Schleife ausgeführt.

Die Schlüsselwörter **while** und **end** müssen als erstes Wort einer Zeile angegeben sein.

Wird die **while**-Schleife interaktiv eingegeben, zeigt die **csH**, solange nicht das abschließende **end** eingegeben wurde, mit der Ausgabe des Prompts ? an, daß die **while**-Schleife noch nicht abgeschlossen ist.

#### Hinweis

Das builtin-Kommando **break** bewirkt, daß die **while**-Schleife verlassen wird. Das builtin-Kommando **continue** bewirkt, daß die Ausführung der *kdos* abgebrochen, und unmittelbar wieder mit der Ausführung des nächsten **while**-Schleifendurchlaufs begonnen wird.

**Beispiel**

Mit den nachfolgenden Kommandozeilen wird allen momentan angemeldeten Benutzern die in der Datei *meldung* enthaltene Nachricht geschickt:

```
% cat meldung␣
Ich starte jetzt fuer eine halbe Stunde
ein sehr rechenintensives Programm.

Ich waere Ihnen sehr dankbar, wenn sie
keine grossen Jobs in dieser Zeit ablaufen
lassen wuerden, damit mein Programm umso
schneller fertig wird.
```

```
Vielen Dank
-- egon
% set argv = `who | cut -d" " -f1`␣
% while ($#argv)␣
? write $1 < meldung␣
? shift␣
? end␣
Ich starte jetzt fuer eine halbe Stunde
ein sehr rechenintensives Programm.
```

```
Ich waere Ihnen sehr dankbar, wenn sie
keine grossen Jobs in dieser Zeit ablaufen
lassen wuerden, damit mein Programm umso
schneller fertig wird.
```

```
Vielen Dank
-- egon
EOT
%
```

Das nachfolgende Shell-Skript *userda* prüft alle 5 Minuten, ob sich ein Benutzer angemeldet hat. Dieses Skript sollte im Hintergrund gestartet werden, um ein Weiterarbeiten während seines Ablaufs zu ermöglichen:

```
% cat userda␣
#!/bin/csh

userda - prueft alle 5 Minuten, ob sich der
Benutzer $1 angemeldet hat.
Erst, wenn sich $1 angemeldet hat,
wird while-Schleife verlassen und
entspr. Meldung ausgegeben.

if ($#argv == 0) then
echo "usage: userda benutzer"
exit(1)
endif

while (! { (who |& grep $1 >& /dev/null) })
```

```
sleep 300
end
echo "$1 hat sich angemeldet"
% chmod u+x userda 
%
```

### 6.13.4 repeat-Schleife

**repeat** *n* *kommando*

Das angegebene *kommando* wird *n* mal ausgeführt. Eventuell angegebene Ein-/Ausgabumlenkungen werden dabei nur einmal vorgenommen und zwar sogar dann, wenn für *n* der Wert **0** angegeben wurde.

Die vollständige **repeat**-Anweisung muß in einer Zeile angegeben sein. Soll z.B. *kommando* in einer eigenen Zeile stehen, muß das Fortsetzungszeichen \ verwendet werden:

**repeat** *n* \  
*kommando*

Für *kommando* muß ein einfaches Kommando angegeben sein. Es darf also keine Pipeline, Kommandoliste oder geklammerte Kommandoliste dafür angegeben werden.

#### Beispiel

Folgender Aufruf kopiert den Inhalt der Datei *datei* fünfmal in die Datei *fuenfdatei*:

```
repeat 5 cat datei >>fuenfdatei
```

### 6.13.5 foreach-Schleife

**foreach** *variable* (*wortliste*)  
*kdos*  
**end**

*variable* wird nacheinander mit den einzelnen Worten der angegebenen *wortliste* besetzt und jedesmal werden die *kdos* ausgeführt.

Die Schlüsselwörter **foreach** und **end** müssen als erstes Wort einer Zeile angegeben sein.

Wird die **foreach**-Schleife interaktiv eingegeben, zeigt die **csH**, solange nicht das abschließende **end** eingegeben wurde, mit der Ausgabe des Prompts ? an, daß die **foreach**-Schleife noch nicht abgeschlossen ist.

#### Hinweis

Das builtin-Kommando **break** bewirkt, daß die **foreach**-Schleife verlassen wird. Das builtin-Kommando **continue** bewirkt, daß die Ausführung der *kdos* abgebrochen, und unmittelbar wieder mit der Ausführung des nächsten **foreach**-Schleifendurchlaufs begonnen wird.



**Beispiel**

Mit den nachfolgenden Aufrufzeilen wird eine Auswertung der am jeweiligen System verwendeten Login-Shells vorgenommen:

```
% foreach i ('sh$' 'csh$' 'ksh$' '-v sh$')
? echo -n "$i "
? grep -c $i /etc/passwd
? end
sh$ 231
csh$ 128
ksh$ 46
-v sh$ 162 [keine Shell angegeben --> sh]
%
```

Das nachfolgende **csh**-Skript *tree* gibt Directorybäume in graphischer Darstellung aus:

```
% cat tree
#!/bin/csh -f

tree - gibt zu einem Directorybaum dessen
Baumstruktur aus

if ($#argv == 0) then
 echo "usage: tree directory"
 exit(1)
endif
if ($?einrueck == 0) setenv einrueck " "
set vorher = "$einrueck"
cd $1
echo "$einrueck `pwd`"
setenv einrueck "$einrueck "
foreach datei (`ls`)
 if (-d $datei) csh -f $home/cshueb/tree $datei
end
setenv einrueck "$vorher"
% chmod u+x tree
% tree /usr
/usr
 /usr/adm
 /usr/adm/acct
 /usr/adm/acct/fiscal
 /usr/adm/acct/nite
 /usr/adm/acct/sum
 /usr/adm/sa
 /usr/adm/streams
 /usr/bin
 /usr/egon
 /usr/games
 /usr/games/doc
 /usr/games/lib
 /usr/games/lib/UTlib
```

```

 /usr/games/lib/Ularn
 /usr/games/lib/gin
 /usr/games/lib/quiz.k
 /usr/games/lib/wanderer
/usr/include
:
:
/usr/preserve
/usr/pub
/usr/spool
 /usr/spool/mail
 /usr/spool/mail/:saved
 /usr/spool/uucp
 /usr/spool/uucp/sosco
 /usr/spool/uucplogins
 /usr/spool/uucplogins/nuucp
 /usr/spool/uucppublic
/usr/sys
 /usr/sys/io
 /usr/sys/mdep
/usr/tmp

```

```
%
```

Das nachfolgende **csh**-Skript *namkill* erlaubt das symbolische Löschen von Prozessen über Namen (nicht wie bei **kill** über die PID):

```
% cat namkill ↵
```

```

#!/bin/csh
#
namkill - Symbolisches Beenden von Prozessen
#
Von der durch ps -ef bewirkten Ausgabe werden mit
fgrep alle Zeilen herausgefiltert, in welchen die
angegebenen Namen enthalten sind.
Die Ausgabe von fgrep wird durch ein zweites
fgrep gefiltert, damit namkill selbst nicht beendet
wird.
Das zweite Feld bei der ps -ef -Ausgabe ist dabei die
PID, welches mit einem awk-Aufruf extrahiert wird.
awk wird zwar erst im naechsten Buch vorgestellt; doch
sollte dieser Aufruf hier selbsterklaerend sein.
#
foreach name($argv)
 set p = `ps -ef | fgrep $name | fgrep -v $0 | awk '{ print $2 }'`
 foreach pid($p)
 kill -9 $pid
 end
end
end
% chmod u+x namkill ↵
%
```

### 6.13.6 goto-Anweisung

**goto** *wort*

bewirkt einen Sprung auf die mit *wort* ausgewählte Marke. Für *wort* wird Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung durchgeführt.

Diese Marke muß wie in C mit

*marke:*

angegeben sein.

```
Beispiel % cat wzaehl ↵
#! /bin/csh
#
wzaehl - zaehlt die Woerter eines Eingabetextes.
Zu Testzwecken wird dabei die
Eingabezeile mit der Anzahl der darin
enthaltenen Woerter nochmals ausgegeben.
Abbruch-Kriterium ist dabei eine
Leerzeile.
#
set noglob
start:
 set zeile = $<
 if ($#zeile == 0) goto ende
 @ sum += $#zeile
 echo "$zeile ($#zeile)"
 goto start

ende:
 echo "Der Eingabetext enthaelt $sum Woerter"
% chmod u+x wzaehl ↵
% wzaehl <ausgab ↵
#!/bin/csh (1)
echo Das erste Argument ist $1 (6)
echo Das zweite Argument ist $2 (6)
echo Der Skriptname ist $0 (5)
Der Eingabetext enthaelt 18 Woerter
%
```

## 6.14 Der History-Mechanismus

Die **csh** ermöglicht es, früher eingegebene Kommandozeilen ganz oder Teile davon wieder zu verwenden. Dabei ist es sogar möglich, über bestimmte Konstruktionen, frühere Kommandozeilen abändern zu lassen, bevor sie wieder zur Ausführung gebracht werden. Dazu bietet die **csh** den sogenannten History-Mechanismus an.

### 6.14.1 Allgemeines

Die **cs**h merkt sich immer das zuletzt eingegebene Kommando. Sollen nicht nur die letzte Kommandozeile, sondern auch noch die davor eingegebenen Kommandozeilen im Kommandopuffer aufgehoben werden, muß die Shell-Variable **history** entsprechend gesetzt werden, z.B.

**set history = 20**

Das Setzen dieser Shell-Variablen sollte in der Datei **.cshrc** (im Home-Directory) erfolgen, damit sie bei jedem Anmelden bzw. Starten einer neuen Subshell automatisch gesetzt wird.

Um sich die Kommandos aus dem History-Puffer anzeigen zu lassen, steht das builtin-Kommando **history** zur Verfügung. **history** kann auf drei verschiedene Arten aufgerufen werden:

- (1) **history**  
listet alle im History-Puffer aufgehobenen Kommandos auf.
- (2) **history n**<sup>1</sup>  
listet von den aufgehobenen Kommandozeilen nur die letzten *n*.
- (3) **history -r [n]**<sup>2</sup>  
listet die aufgehobenen Kommandozeilen in umgekehrter Reihenfolge. Ist *n* angegeben, werden von den aufgehobenen Kommandozeilen nur die letzten *n* in umgekehrter Reihenfolge ausgegeben.

In der **cs**h werden die einzelnen Kommandozeilen als *events*<sup>3</sup> bezeichnet und fortlaufend numeriert. Bei der Ausgabe mit **history** werden die Nummern der einzelnen Kommandozeilen (*event numbers*) mit ausgegeben. Um sich immer die aktuelle event number der momentanen Kommandozeile anzeigen zu lassen, muß das Zeichen **!** im Wert der Shell-Variablen **prompt** angegeben werden.

---

1. Auf manchen **cs**h-Versionen nicht verfügbar.  
2. Auf manchen **cs**h-Versionen nicht verfügbar.  
3. Zu deutsch: Ereignisse.

## 6.14.2 History-Substitutionen

Soll auf eine zuvor eingegebene Kommandozeile wieder zugegriffen werden, muß das Ausrufezeichen ! angegeben werden.

Nach dem Ausrufezeichen kann nun folgendes angegeben werden:

| Angabe                  | Bedeutung                                                                                                                                                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ! <i>n</i>              | Hierfür wird die Kommandozeile mit der event number <i>n</i> in der aktuellen Kommandozeile eingesetzt; z.B. würde für <b>!12</b> die Kommandozeile mit event number 12 eingesetzt.                                                                                  |
| ! <i>n</i>              | Hierfür wird die <i>n</i> .te vorherige Kommandozeile in der aktuellen Kommandozeile eingesetzt; wenn z.B. die aktuelle Zeile die event number 123 hätte, würde für <b>!-4</b> die Kommandozeile mit der event number 119 eingesetzt.                                |
| !!                      | Hierfür wird die unmittelbar zuvor eingegebene Kommandozeile in der aktuellen Kommandozeile eingesetzt; !! entspricht also <b>!-1</b> .                                                                                                                              |
| ! <i>string</i>         | Es wird rückwärts in den aufgehobenen Kommandozeilen nach einer Zeile gesucht, die mit <i>string</i> beginnt. Wird eine solche gefunden, dann wird sie hierfür eingesetzt.                                                                                           |
| ! <i>?string?</i>       | Es wird rückwärts in den aufgehobenen Kommandozeilen nach einer Zeile gesucht, die <i>string</i> enthält. Wird eine solche gefunden, dann wird diese hierfür eingesetzt.                                                                                             |
| ! <i>#</i> <sup>a</sup> | Hierfür wird die bisher eingegebene aktuelle Zeile eingesetzt. Mit dieser Aufrufform ist es möglich, auf zuvor (weiter links) eingegebene Wörter wieder zuzugreifen, ohne diese erneut eingeben zu müssen. Dies ist vor allen Dingen bei langen Wörtern von Vorteil. |

a. Wird nicht auf allen **csh**-Versionen angeboten.

Bei allen diesen Konstruktionen wird eine Kommandozeile nur dann gefunden, wenn sie sich noch im Kommandopuffer befindet, also zuvor maximal **\$history** Zeilen eingegeben wurden.

Die Sonderbedeutung von ! kann mit einem vorangestellten \ ausgeschaltet werden. Das Ausrufezeichen verliert allerdings auch dann seine Sonderbedeutung für den History-Mechanismus, wenn ihm ein Leerzeichen, Tabulatorzeichen, Neuezeile-Zeichen, = oder ( folgt.

Nun ist es jedoch auch möglich, daß man nicht die ganze Kommandozeile, sondern nur Teile davon einsetzen läßt. Dazu muß die folgende Konstruktion verwendet werden:

`langabe[:auswahl][:modifikator(en)]`

Für *langabe* muß eine der sechs zuvor beschriebenen Formen verwendet werden.

Für *auswahl* kann dabei folgendes angegeben werden:

| Auswahl    | Wortauswahl in entsprechender Kommandozeile                                                                                                     |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>n</i>   | <i>n</i> .te Wort.<br><br>Dabei ist zu beachten, daß <b>0</b> das erste Wort, <b>1</b> das zweite Wort, <b>2</b> das dritte Wort usw. auswählt. |
| <b>^</b>   | erste Argument (nicht Kommandoname)                                                                                                             |
| <b>\$</b>  | letztes Argument                                                                                                                                |
| <b>%</b>   | das durch die zuvor angegebene Konstruktion <i>?string?</i> ausgewählte Wort                                                                    |
| <i>m-n</i> | <i>m</i> bis <i>n</i> .te Wort.<br><br>Wortzählung beginnt wieder bei 0.                                                                        |
| <i>-n</i>  | entspricht <b>0-n</b>                                                                                                                           |
| <i>m-</i>  | <i>m</i> bis vorletzte Wort                                                                                                                     |
| <b>*</b>   | entspricht <b>^-\$</b>                                                                                                                          |
| <i>n*</i>  | entspricht <i>n-\$</i>                                                                                                                          |

Der Doppelpunkt : vor *auswahl* kann weggelassen werden, wenn *auswahl* mit **^**, **\$**, **\***, **-** oder **%** beginnt.

Die über Angabe von *auswahl* ausgewählten Wörter könnten durch Angabe von *modifikatoren* noch modifiziert werden. Als *modifikatoren* sind dabei möglich:

| Modifikator       | Auswirkung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>h</b>          | ( <b>head</b> ) liefert den Pfadnamen des entsprechenden Worts.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>t</b>          | ( <b>tail</b> ) liefert den Basisnamen des entsprechenden Worts (Gegenstück zu <b>h</b> ).                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>r</b>          | ( <b>remove</b> ) schneidet vom entsprechenden Wort eine vorhandene Endung .xxx ab.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>e</b>          | ( <b>extension</b> ) liefert vom entsprechenden Wort nur eine eventuell vorhandene Endung .xxx.                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>s/alt/neu/</b> | ( <b>substitute</b> ) ersetzt den durch <i>alt</i> abgedeckten Text durch <i>neu</i> . Für <i>alt</i> dürfen die bei <b>ed</b> zugelassenen regulären Ausdrücke <sup>a</sup> verwendet werden.<br><br>Wird das Zeichen <b>&amp;</b> in <i>neu</i> angegeben, wird hierfür der durch <i>alt</i> abgedeckte Text eingesetzt.<br><br>Wird für <i>neu</i> keinerlei Text angegeben, wird dafür das <i>neu</i> aus einem vorherigen <b>s/alt/neu/</b> oder aus einer vorherigen Angabe <i>!?string?</i> genommen. |

| Modifikator  | Auswirkung                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | Statt / darf jedes beliebige Trennzeichen verwendet werden, solange es nicht in <i>alt</i> oder <i>neu</i> vorkommt.<br><br>Wenn nach <b>s/alt/neu/</b> sofort ein Neuezeile-Zeichen angegeben wird, darf das letzte / auch weggelassen werden.<br><br>Um die Bedeutung von Sonderzeichen wie <b>&amp;</b> oder / in <i>alt</i> oder <i>neu</i> auszuschalten, muß diesen ein \ vorangestellt werden. |
| <b>&amp;</b> | wiederholt die letzte mit <b>s/alt/neu/</b> angegebene Substitution.                                                                                                                                                                                                                                                                                                                                  |
| <b>g</b>     | ( <b>g</b> lobal) entsprechender Modifikator wird nicht nur auf das erste Wort, sondern auf alle ausgewählten Wörter angewendet. <b>g</b> ist dabei vor dem entsprechenden Modifikator ( <b>h</b> , <b>r</b> , <b>e</b> , <b>s</b> , <b>t</b> , <b>&amp;</b> ) anzugeben.                                                                                                                             |
| <b>p</b>     | ( <b>p</b> rint) die neu entstandene Kommandozeile wird nur angezeigt, aber nicht ausgeführt.                                                                                                                                                                                                                                                                                                         |
| <b>q</b>     | ( <b>q</b> uote) für die entsprechende Kommandozeile wird Quoting vorgenommen, so daß keine weiteren Substitutionen oder Dateinamen-Expandierungen stattfinden.                                                                                                                                                                                                                                       |
| <b>x</b>     | wie <b>q</b> , nur daß der entsprechende Wert in mehrere Worte aufgeteilt wird. Trennzeichen sind dabei Leer-, Tabulator- und Neuezeile-Zeichen.                                                                                                                                                                                                                                                      |

a. Siehe »Linux-Unix-Grundlagen«.

Nur wenn **g** als erster Modifikator angegeben ist, werden die Modifikationen für alle Wörter durchgeführt, ansonsten werden sie nur für das erste Wort vorgenommen.

Wird keine *angabe* gemacht, bezieht sich die entsprechende Konstruktion immer auf die vorherige Zeile, z.B. entspricht **!\$** der Konstruktion **!!:\$. Dies gilt jedoch auf manchen **cs**h-Versionen nur, wenn nicht bereits durch eine zuvor in dieser Zeile angegebene Konstruktion eine andere Zeile ausgewählt wurde, z.B. liefert **!35:^ !\$** das erste und letzte Argument der Kommandozeile mit event number 35.**

Die spezielle Angabe **^alt^neu^** ersetzt in der vorherigen Kommandozeile den durch *alt* abgedeckten Text durch den Text *neu*. So ist eine einfache Korrektur von Tippfehlern möglich, ohne daß die vollständige Form:

**!!:s/alt/neu/**

angegeben werden muß.

Nachdem die History-Substitutionen durchgeführt wurden, wird die entsprechende Kommandozeile vor ihrer Ausführung angezeigt.

```
Beispiel % set history = 30
% set prompt="! % "
12 % echo eins zwei drei vier add.c subtr.pas
eins zwei drei vier add.c subtr.pas
13 % echo !!:2-3
echo zwei drei
zwei drei
14 % !!
echo zwei drei
zwei drei
15 % find /usr -name stdio.h -print
/usr/include/stdio.h
16 % !12:0 !12:4-6
echo vier add.c subtr.pas
vier add.c subtr.pas
17 % !!:r
echo vier add subtr.pas
vier add subtr.pas
18 % !find:s/stdio/ctype
find /usr -name ctype.h -print
/usr/include/ctype.h
19 % echo Hallo Franz, wie gehts Franz
Hallo Franz, wie gehts Franz
20 % !!:s/Franz/Emil
echo Hallo Emil, wie gehts Franz
Hallo Emil, wie gehts Franz
21 % !19:gs/Franz/Fritz
echo Hallo Fritz, wie gehts Fritz
Hallo Fritz, wie gehts Fritz
22 % cyt add.c
cyt: command not found.
23 % ^y^a
cat add.c
main()
{
 int a, b;
 printf("Gib 2 Zahlen ein: ");
 scanf("%d %d", &a, &b);
 printf("Summe ist : %d\n", a+b);
}
24 % cc add.c
add.c
add.c (4) : error ...: newline in constant
add.c (5) : error: syntax error
add.c (6) : error ...: newline in constant
25 % ed add.c
118
P
*,n
1 main()
2 {
```



```

3 int a, b;
4 printf("Gib 2 Zahlen ein: ");
5 scanf("%d %d", &a, &b);
6 printf("Summe ist : %d\n", a+b);
7 }
*4s//")
 printf("Gib 2 Zahlen ein: ");
*w
119
*q
26 % !cc
cc add.c
add.c
add.c (5) : error syntax error
add.c (5) : error 'd' y. undefined
add.c (5) : error newline in constant
.....
27 % !ed
ed add.c
119
P
*,n
1 main()
2 {
3 int a, b;
4 printf("Gib 2 Zahlen ein: ");
5 scanf("%d %d", &a, &b);
6 printf("Summe ist : %d\n", a+b);
7 }
*5s//("
 scanf("%d %d", &a, &b);
*w
120
*q
28 % !cc -o add
cc add.c -o add
add.c
29 % file !!^ !!$
file add.c add
add.c: c program text
add: ...executable not stripped
30 % ls !!*
ls add.c add
add
add.c
31 % echo ----- Ueberschrift !#:1
echo ----- Ueberschrift -----
----- Ueberschrift -----
32 % echo !$
echo -----

```

```
33 % history[↵]
.....
11 set prompt ="! % "
12 echo eins zwei drei vier add.c subtr.pas
13 echo zwei drei
14 echo zwei drei
15 find /usr -name stdio.h -print
16 echo vier add.c subtr.pas
17 echo vier add subtr.pas
18 find /usr -name ctype.h -print
19 echo Hallo Franz, wie gehts Franz
20 echo Hallo Emil, wie gehts Franz
21 echo Hallo Fritz, wie gehts Fritz
22 cyt add.c
23 cat add.c
24 cc add.c
25 ed add.c
26 cc add.c
27 ed add.c
28 cc add.c -o add
29 file add.c add
30 ls add.c add
31 echo ----- Ueberschrift -----
32 echo -----
33 history
34 %
```

**Hinweis**

Die *angabe* kann auch mit `{..}` geklammert werden. Wenn z.B. eine der vorherigen Zeile mit folgenden Inhalt:

**echo Hallo emil**

wieder auszuführen ist, allerdings mit

**echo Hallo emilia**

dann würde die Angabe **!eia** nach einer Zeile suchen, die mit **eia** anfängt. In diesem Fall wäre dann **!(e)ia** anzugeben.

Kommandos innerhalb **while**-, **foreach**- oder **if**-Kommandos, die beim Promptzeichen **?** eingegeben werden, werden nicht im History-Puffer aufgehoben.

## 6.15 Der Alias-Mechanismus

Mit dem Alias-Mechanismus der **csh** ist es möglich, an einen ganzen oder auch unvollständigen Kommandoaufruf einen Kurznamen, auch *Alias* genannt, zu vergeben. Der Alias-Mechanismus kann also dazu verwendet werden, um Kürzel für Kommandos einzuführen oder Kommandos mit default-Argumenten zu versorgen oder neue, kurze Kommandonamen für ganze Kommandozeilen einzuführen, in denen Pipelines und Ein-/Ausgabeumlenkungen vorkommen.

### 6.15.1 Das builtin-Kommando alias

Mit dem builtin-Kommando **alias** können entweder Aliase neu definiert oder aber momentan definierte Aliase angezeigt werden. **alias** kann auf drei verschiedene Arten aufgerufen werden:

#### **alias**

gibt alle momentan definierten Aliase aus.

#### **alias name**

gibt den Wert des Alias *name* aus, wenn ein solches existiert.

#### **alias name wortliste**

definiert ein Alias *name*: Dem Alias *name* wird der Wert *wortliste* zugewiesen. Für jedes einzelne Wort der *wortliste* wird dabei Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung durchgeführt.

Als Aliasname darf weder *alias* noch *unalias* gewählt werden.

Wenn ein Alias-Text immer dann zu expandieren ist, wenn auf das entsprechende Alias zugegriffen wird, muß der Alias-Text mit `'..'` geklammert werden, ansonsten wird für den Alias-Text bereits bei der Definition Dateinamen-Expandierung, Parametersubstitution und Kommandosubstitution durchgeführt, was nicht immer erwünscht ist.

Ebenso muß ein Alias-Text mit `'..'` geklammert werden, wenn in ihm **cs**h-Metazeichen wie `;`, `>` usw. vorkommen.

#### Beispiel

```
% alias dir 'ls -l'
% alias type cat
% alias h history
% alias ll ls -CF
% alias
dir ls -l
h history
ll (ls -CF)
type cat
% alias dir
ls -l
% alias dd
% set x = 1
% alias druck1 'echo $x'
% alias druck2 echo $x
% set x = 5
% druck1
5
% druck2
1
%
```

## 6.15.2 Alias-Substitution

Für jedes einfache Kommando prüft die **cs**h das erste Wort (Kommandoname), um dieses eventuell als Alias zu klassifizieren. Wenn keinerlei Quoting in diesem Wort vorkommt und das Wort ein erlaubter Alias-Name ist, dann setzt die **cs**h für den Alias, falls definiert, den entsprechenden Alias-Text ein.

**Beispiel**

```
% alias drucke echo
% alias text 'Hallo egon'
% drucke text
text
% alias gross "tr '[a-z]' '[A-Z]'"
% cat ausgab | gross
!#/BIN/CSH
ECHO DAS ERSTE ARGUMENT IST $1
ECHO DAS ZWEITE ARGUMENT IST $2
ECHO DER SKRIPTNAME IST $0
%
```

Die so entstandene neue Kommandozeile wird von der **cs**h erneut von Beginn an gelesen. Kommen im Alias-Text History-Konstruktionen vor, beziehen sie sich auf die ursprünglich gegebene Zeile, bevor Alias-Substitution durchgeführt wurde.

**Beispiel**

```
% alias usuch 'grep \!^ /etc/passwd'
% usuch emil
emil:zgusa63kszuaw78:201:50:Emil Lerner:/user1/emil:/bin/sh
% set echo
% usuch emil
grep emil /etc/passwd
emil:zgusa63kszuaw78:201:50:Emil Lerner:/user1/emil:/bin/sh
% unset echo
unset echo
%
```

Der Aufruf von

**usuch emil**

resultiert in folgender Kommandozeile

**grep \!^ /etc/passwd**

Diese Zeile wird von der **cs**h erneut gelesen, wobei nun für **\!^** das erste Argument aus der ursprünglichen Kommandozeile (**emil**) eingesetzt wird, so daß schließlich folgende endgültige Kommandozeile entsteht:

**grep emil /etc/passwd**

Falls nach einer Alias-Substitution ein weiteres Alias (als erstes Wort) entsteht, beginnt die Alias-Substitution für dieses Wort erneut:

```
% alias liste dir
% alias dir ls -CF
% liste [a-e]*
add* add.c ausgab* copy* eingabe*
% set echo
% !!i
liste [a-e]*
ls -CF add add.c ausgab copy eingabe
add* add.c ausgab* copy* eingabe*
% unset echo
unset echo
%
```

Nach der 1. Alias-Substitution für

**liste [a-e]\***

ergab sich folgende Kommandozeile

**dir [a-e]\***

was zu einer erneuten Alias-Substitution führte, so daß schließlich folgende endgültige Kommandozeile entstand:

**ls -CF [a-e]\***

Kommt in einem Alias-Text der Alias-Name selbst nochmals vor, wird dieser nicht erneut substituiert, um eine endlose Alias-Substitution zu unterbinden:

```
% alias ls ls -CF
% ls t*
text1 tree* typ* typ2*
%
```

Diese Vorgehensweise der **csh** ermöglicht es, Alias-Namen von Kommandos zu verwenden, um die default-Einstellung von Kommandos zu ändern, wie z.B.

```
% alias rm rm -i
% rm t*
text1: ?
tree: ?
typ: ?
typ2: ?
% alias cd 'cd \!^:pwd'
% cd /usr
/usr
% cd ~/cshueb
/user1/egon/cshueb
%
```

### 6.15.3 Löschen eines Alias

Mit dem builtin-Kommando **unalias** können zuvor definierte Aliase gelöscht werden:

**unalias** *pattern* ...

Es werden dabei alle Aliase, die durch die angegebenen *pattern* abgedeckt werden, gelöscht. Im *pattern* sind dabei die Metazeichen **\***, **?** und **[.]** erlaubt. Mit dem Aufruf

**unalias** **\***

werden z.B. alle Aliase gelöscht. Falls die angegebenen *pattern* keinen einzigen Aliasnamen abdecken, ist dies kein Fehler.

## 6.16 Fehlersuche in csh-Skripts

Die **csh** bietet folgende Optionen zum Debuggen von Skripts an:

| Optionen  | Bedeutung                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-n</b> | ( <b>noexec</b> ) Kommandos werden nur gelesen und auf Syntaxfehler untersucht, aber nicht ausgeführt. Diese Option kann für eine syntaktische Überprüfung von <b>csh</b> -Skripts verwendet werden.                                                                                                                                                                                         |
| <b>-v</b> | ( <b>verbose</b> ) bewirkt, daß die Shell-Variable <b>verbose</b> gesetzt wird, so daß nach jeder History-Substitution die neu entstandene Kommandozeile vor ihrer Ausführung angezeigt wird.                                                                                                                                                                                                |
| <b>-V</b> | ( <b>Verbose</b> ) entspricht der Option <b>-v</b> , nur daß die Shell-Variable <b>verbose</b> bereits vor der Ausführung von <b>.cshrc</b> gesetzt wird.                                                                                                                                                                                                                                    |
| <b>-x</b> | ( <b>xtrace</b> ) bewirkt, daß die Shell-Variable <b>echo</b> gesetzt wird, so daß jede Kommandozeile vor ihrer Ausführung nochmals ausgegeben wird. Während jedoch bei nicht builtin-Kommandos alle Substitutionen und Expandierungen vor dieser Ausgabe bereits durchgeführt sind, werden bei builtin-Kommandos diese Expandierungen und Substitutionen erst nach der Ausgabe vorgenommen. |
| <b>-X</b> | ( <b>Xtrace</b> ) entspricht der Option <b>-x</b> , nur daß die Shell-Variable <b>echo</b> bereits vor der Ausführung von <b>.cshrc</b> gesetzt wird.                                                                                                                                                                                                                                        |

Diese Optionen können beim Aufruf eines **cs****h**-Skripts gesetzt werden:

**cs****h** [-**nvVxX**] *skript*

In diesem Fall gelten die gesetzten Optionen nur für die Dauer der Skript-Ausführung.

Das Setzen der Optionen **-V** und **-X** kann jedoch auch indirekt mit

**set verbose** oder

**set echo**

in der gerade aktiven **cs****h** bzw. im entsprechenden **cs****h**-Skript erreicht werden.

## 6.17 Signalbehandlung in der cs

In der **cs****h** ist die Signalbehandlung nicht so komfortabel wie in der Bourne-Shell. Es können nämlich in der **cs****h** nur die folgenden Signale abgefangen werden:

| Angabe           | Beschreibung                                                                                                      |
|------------------|-------------------------------------------------------------------------------------------------------------------|
| <i>intr</i>      | Interrupt-Signal, welches durch Drücken der <i>DEL</i> - oder <i>BREAK</i> -Taste ( <i>Strg-c</i> ) erzeugt wird. |
| <i>hangup</i>    | Wird beim Beenden einer Verbindung (z.B. Auflegen des Telefonhörers) erzeugt.                                     |
| <i>terminate</i> | Ist das voreingestellte Signal für das builtin-Kommando <b>kill</b> .                                             |

Die Signalnummern für diese Signale sind systemabhängig. Mögliche Signalnummern für diese Signale können in Kapitel 4.12.2 nachgeschlagen werden.

Zum Abfangen der obigen Signale steht das builtin-Kommando **onintr** zur Verfügung, das auf drei verschiedene Arten aufgerufen werden kann:

**onintr**

stellt die default-Signalbehandlung der **cs****h** wieder ein, die beim Eintreffen der obigen Signale das Beenden des entsprechenden Shell-Skripts bedeutet.

**onintr -**

bewirkt, daß die obigen Signale ignoriert werden.

**onintr marke**

bewirkt, daß beim Auftreten der oben angegebenen Signale in der aktuellen **cs****h** oder in Kindprozessen (zu dieser **cs****h**) die Marke *marke* angesprungen wird.

Bei Hintergrund-Jobs werden alle Unterbrechungs- und Beendigungs-Signale ignoriert, unabhängig davon, was mit **onintr** als Signalbehandlung eingestellt wurde.

```
Beispiel % cat delfang ↵
#!/bin/csh
zaehlt von 1 bis $1 hoch.
Ist $1 nicht angegeben, so wird 1000 dafuer
angenommen.
#
Wird waehrend der Berechnung eine Unterbrechungstaste
gedruickt, so wird die bis dahin berechnete Zahl
ausgegeben.

onintr sighandler

if ($#argv == 0) set argv = 1000
set x=1

weiter:
 while ($x <= $argv[1])
 @ x = $x + 1
 end
 echo $x
 exit 0

sighandler:
 onintr
 echo $x
 onintr sighandler
 goto weiter
% chmod u+x delfang ↵
% delfang ↵
[DEL]235 [eventuell muß DEL auch mehrmals gedrückt werden]
[DEL]461
[DEL]769
1001
%
```

**Hinweise** Die mit **onintr** installierten Signalhandler werden an Subshells vererbt.

Während des Lesens von *.logout* durch die Login-Shell sind Unterbrechungs-Signale verboten.

Das *quit*-Signal (*Strg-\*) kann normalerweise in der **csh** nicht abgefangen werden.

Typische Anwendungen:

Wenn für einen bestimmten kritischen Abschnitt in einem **csh**-Skript Unterbrechungen nicht gestattet werden sollen, muß dieser Abschnitt wie folgt geklammert werden:

```
onintr - # Interrupts ignorieren
.....
..... # kritischer, nicht unterbrechbarer Abschnitt
```



```
.....
onintr # Interrupts wieder zulassen
```

Löschen von temporären Dateien beim Beenden eines **csh**-Skripts durch Interrupts:

```
set tmpdatei = /tmp/....$$
set exit_wert = 1
onintr sighandler

.....
..... # eigentliches csh-Skript
.....
onintr -
set exit_wert = 0

sighandler:
/bin/rm -f $tmpdatei >/dev/null
exit $exit_wert
```

## 6.18 Job-Kontrolle in der csh

Jede Pipeline<sup>1</sup>, die man ausführen läßt, wird als Job bezeichnet. Manche **csh**-Versionen, jedoch nicht alle, verfügen anders als die Bourne-Shell über eine eigene Job-Kontrolle, mit der es möglich ist, Jobs anzuhalten und wieder fortzusetzen, oder die Ausführung von Jobs vom Hintergrund in den Vordergrund zu verlagern bzw. umgekehrt.

### 6.18.1 Allgemeines

Bei jedem Start eines Hintergrund-Jobs gibt die **csh** dessen Jobnummer mit [...] geklammert gefolgt von den PIDs der zu diesem Jobs gehörigen Prozesse aus.

Wenn die Ausführung eines Hintergrund-Jobs beendet ist, meldet dies die **csh** vor der Ausgabe des nächsten Prompts. Aus der Meldung läßt sich schließen, ob der Job normal beendet wurde (*Done*) oder nicht (*Killed*, *Illegal Instruction* usw.).

#### Beispiel

```
% find /user1 -print | wc -l & ↵
[1] 1378 1379
% pwd ↵
/user1/egon/cshueb
3567 [Anzahl von Dateien im Directorybaum /user1]
% pwd ↵
/user1/egon/cshueb
[1] Done find /user1 -print | wc -l
%
```

---

1. Als Erinnerung: Eine Pipeline kann aus nur einem Kommando bestehen.

Soll die Meldung über die Beendigung eines Hintergrund-Jobs sofort und nicht erst vor der Ausgabe des nächsten Prompts erfolgen, muß die Shellvariable **notify** gesetzt werden.

**Beispiel**

```
% ls /bin | wc -l & ↵
[1] 1422 1423
 145
% pwd ↵
/user1/egon/cshueb
[1] Done ls /bin | wc -l
% set notify ↵
% ls /bin | wc -l & ↵
[1] 1428 1429
 145
[1] Done ls /bin | wc -l
%
```

Die einzelnen Jobs können über die PID, die Jobnummer oder den Jobnamen angesprochen werden. Dazu stehen die builtin-Kommandos **wait**, **kill**, **fg**, **bg**, **stop**, **suspend** und **notify** zur Verfügung. Um einen Job über seine Jobnummer oder seinen Namen anzusprechen, gibt es folgende Notationen:

| Angabe        | ausgewählter Job                                                                                                            |
|---------------|-----------------------------------------------------------------------------------------------------------------------------|
| %jobnr        | Job mit Jobnummer <i>jobnr</i>                                                                                              |
| %string       | Job, dessen Name mit <i>string</i> beginnt; <i>string</i> kann dabei die Metazeichen der Dateinamen-Expandierung enthalten. |
| %?string      | Job, dessen Name den <i>string</i> enthält; <i>string</i> kann dabei die Metazeichen der Dateinamen-Expandierung enthalten. |
| %+ oder<br>%% | aktueller Job                                                                                                               |
| %-            | vorheriger aktueller Job                                                                                                    |

Jobs werden eingeteilt in Vordergrund-Jobs, Hintergrund-Jobs und suspendierte (angehaltene) Jobs. Nur ein Job kann zu einem Zeitpunkt im Vordergrund arbeiten, aber es können mehrere Jobs gleichzeitig im Hintergrund ablaufen oder suspendiert sein.

Mit dem builtin-Kommando **jobs** können die gerade im Hintergrund ablaufenden und angehaltenen Jobs mit ihrem derzeitigen Status angezeigt werden.

Der Job, der gerade im Vordergrund abgearbeitet wird, kann mit der *Suspend*-Taste (meist *Strg-z*)<sup>1</sup> angehalten werden. Die **csh** meldet dann mit der Ausgabe »*Stopped*«, daß der entsprechende Job angehalten wurde und zeigt mit der Aus-

1. Kann auch mit **stty** eingestellt werden, z.B. **stty susp '^Z'**.

gabe des Prompts an, daß sie nun wieder für die Entgegennahme von Kommandos bereit ist. Suspendieren (Anhalten) von Vordergrund-Prozessen ist immer dann notwendig, wenn eine momentane Aktivität kurz unterbrochen werden muß, um andere Kommandos ausführen zu lassen. Nachdem die anderen Kommandos ausgeführt wurden, kann der angehaltene Job mit dem builtin-Kommando **fg** wieder in den Vordergrund gebracht werden.

**Beispiel**

```
% mail emil ↵
Hallo Emil ↵
↵
Ich habe ein csh-Skript fuer Fibonacci-Zahlen ↵
Der Name ist: ↵
[Strg-z]
Stopped [Meldung, dass mail-Programm angehalten]
% ls ~/cshueb/f* ↵
/user1/egon/cshueb/fiba
% fg ↵ [setzt Ausführung von mail fort]
mail emil [Meldung, dass nun Ausführung von mail fortgesetzt wird]
/user1/egon/cshueb/fiba ↵
. ↵
%
```

Vordergrund-Jobs können auch mit der *suspend*-Taste angehalten werden, um ihre Ausführung mit dem builtin-Kommando **bg** im Hintergrund fortsetzen zu lassen. Dies ist besonders dann nützlich, wenn ein Vordergrund-Job länger für die Ausführung benötigt als erwartet. So muß dieser nicht vollständig abgebrochen werden, sondern kann mit seiner Ausführung am unterbrochenen Punkt fortfahren.

**Beispiel**

```
% find / -print | wc -l ↵ [Alle Dateien des Dateisystems zählen]
[Strg-z]
Stopped
% bg ↵
[1] find / -print | wc -l &
%
```

Die *suspend*-Taste (*Strg-z*) sollte nur als erstes Zeichen einer Zeile eingegeben werden, da der ganze zuvor eingetippte Text in diesem Fall weggeworfen wird. Das gleiche gilt im übrigen auch für *intr* und *quit*.

Ein Hintergrund-Job kann mit dem builtin-Kommando **stop** angehalten werden.

Ein Hintergrund-Job wird immer dann automatisch angehalten, wenn er versucht, vom Terminal zu lesen. Die **csh** meldet dies mit

```
[jobnr] Stopped (tty input) jobname
```

Mit dem Kommando

### **stty tostop**

kann zusätzlich noch festgelegt werden, daß Hintergrund-Jobs auch dann anzuhalten sind, wenn sie versuchen, auf das Terminal zu schreiben. Die **cs**h meldet dies mit

```
[jobnr] Stopped (tty output) jobname
```

So können störende Ausgaben von Hintergrund-Jobs unterbunden werden.

Erst wenn die betreffenden Hintergrund-Jobs mit **fg** in den Vordergrund gebracht werden, können sie dann ihre Ein- bzw. Ausgaben durchführen.

Nachfolgend werden nun noch die für die Job-Kontrolle benötigten builtin-Kommandos genauer vorgestellt.

## **6.18.2 Informationen zu Hintergrund-Jobs und angehaltenen Jobs (jobs)**

### **jobs [-l]**

Mit dem builtin-Kommando **jobs** können Informationen zu allen momentan vorhandenen Hintergrund-Jobs und angehaltenen Jobs ausgegeben werden.

Die **cs**h gibt dabei zu jedem einzelnen Job eine Zeile aus, in der folgende Information enthalten ist:

[Jobnummer]            mit einem + vor bzw. nach dem aktuellen Job und einem – (Minuszeichen) vor bzw. nach dem vorherigen aktuellen Job.

Status                *Running*            befindet sich in der Ausführung

*Stopped*            ist momentan angehalten

*Done*                wurde normal beendet

*Terminated*       wurde abgebrochen

Kommandozeile        gibt die Kommandozeile wieder, die beim Start des Jobs angegeben wurde.

### **Optionen**

**-l**                      nach der Jobnummer wird zusätzlich noch die PID des Jobs ausgegeben.

Da das Kommando **jobs** nur Jobs ausgibt, die in der aktuellen **cs**h gestartet wurden, kennt es keine Hintergrund-Jobs aus anderen Login-Sitzungen. Um Informationen über diese zu erhalten, müßte das Kommando **ps** verwendet werden.

```
Beispiel % jobs -l 
+[4] 1329 Running cc -c mult.c &
-[3] 1452 Stopped mail eml
[2] 1632 Done find / -name "*.h" | wc -l &
%
```

### 6.18.3 Signale an Jobs schicken (kill)

**kill** [-signal] job(s) (1) oder

**kill -l** (2)

Mit dem Kommando **kill** wird den angegebenen *job(s)* (Hintergrund-Jobs oder angehaltene Jobs) das Signal *signal* geschickt. Es bewirkt den Abbruch der entsprechenden *job(s)*, wenn es von ihnen nicht ignoriert oder abgefangen wird.

Für *job(s)* können auch PIDs von Prozessen angegeben werden.

Für *signal* kann entweder eine Signalnummer oder ein Signalname angegeben werden. Als Signalname sind dabei die in der Header-Datei */usr/include/sys/signal.h* definierten Namen ohne das Präfix "SIG" zu verwenden.

Ist *-signal* nicht angegeben, dann wird das Signal **TERM** geschickt.

Falls einem angehaltenen Job das Signal **TERM** oder **HUP** geschickt wird, sendet die **csh** diesem Job zuvor das Signal **CONT**.

Die **csh** meldet die durch **kill** bedingte Beendigung eines Jobs mit

[jobnr] Terminated jobname

Um Hintergrund-Jobs anzuhalten, steht das builtin-Kommando **stop** zur Verfügung (siehe auch Kapitel 6.18.7). Diese Kommando könnte auch mit:

**kill -STOP ...**

nachgebildet werden.

Um sich alle auf dem jeweiligen System verfügbaren Signalnummern und Signalnamen anzeigen zu lassen, steht die zweite Aufrufform (2) zur Verfügung:

**kill -l**

```
Beispiel % find / -name "*.c" -print | wc -l & 
[1] 1422 1423
% kill %1 
[1] Terminated find / -name "*.c" -print | wc -l
%
```

### 6.18.4 Ausführung von angehaltenen Jobs im Hintergrund fortsetzen (**bg**)

**bg** [*job(s)*]

Das builtin-Kommando **bg** bewirkt, daß die Ausführung der angehaltenen *job(s)* im Hintergrund fortgesetzt wird. Werden keine *job(s)* beim Aufruf angegeben, wird die Ausführung des aktuellen Jobs (zuletzt angehaltener Job) im Hintergrund fortgesetzt. Der aktuelle Job kann auch mit dem Kommando **jobs** ermittelt werden.

### 6.18.5 Ausführung von angehaltenen Jobs und Hintergrund-Jobs im Vordergrund fortsetzen (**fg**)

**fg** [*job(s)*]

Das builtin-Kommando **fg** bewirkt, daß die Ausführung der angegebenen *job(s)* (Hintergrund-Jobs und angehaltene Jobs) der Reihe nach im Vordergrund fortgesetzt wird. Werden keine *job(s)* beim Aufruf angegeben, wird die Ausführung des aktuellen Jobs (zuletzt angehaltener Job oder zuletzt im Hintergrund gestarteter Job) im Vordergrund fortgesetzt. Der aktuelle Job kann auch mit dem Kommando **jobs** ermittelt werden.

#### Hinweis

Auf manchen *cs*h-Versionen kann **fg** auch weggelassen und nur *job(s)* angegeben werden.

### 6.18.6 Auf die Beendigung von Jobs warten (**wait**)

**wait**

Das builtin-Kommando **wait** veranlaßt die **cs**h auf die Beendigung aller Hintergrund-Prozesse zu warten.

Wenn die **cs**h interaktiv ist, kann ein Interrupt ein **wait** unterbrechen; in diesem Fall gibt die **cs**h alle Job-Namen mit Job-Nummern aus, die noch nicht beendet sind.

#### Beispiel

```
% date; sleep 10 &; wait; date ↵
1492
Tue Jul 30 15:08:06 MESZ 1997
Tue Jul 30 15:08:16 MESZ 1997
% date; sleep 10 &; date ↵
Tue Jul 30 15:09:17 MESZ 1997
1582
Tue Jul 30 15:09:17 MESZ 1997
%
```

## 6.18.7 Anhalten von Hintergrund-Jobs (stop)

**stop** [*job(s)*]

Das builtin-Kommando **stop** bewirkt, daß die Ausführung der angegebenen Hintergrund-*job(s)* angehalten wird. Werden keine *job(s)* beim Aufruf angegeben, wird die Ausführung des aktuellen Jobs (zuletzt im Hintergrund gestarteter Job) angehalten. Der aktuelle Job kann auch mit dem Kommando **jobs** ermittelt werden.

Der Aufruf von

**stop** *job(s)* ist identisch zu **kill -STOP** *job(s)*

### Beispiel

```
% find / -name "*.c" -print | wc -l &
[1] 1422 1423
% stop %1
[1] Stopped (signal) find / -name "*.c" -print | wc -l
%
```

### Hinweise

Wenn die Ausführung eines angehaltenen Jobs wieder fortgesetzt wird, dann gibt die **cs**h das Directory aus, in dem der Job gestartet wurde. Dies kann irreführend sein, da der betreffende Job eventuell bereits intern einen Directorywechsel vorgenommen hat.

Builtin-Kommandos können nicht angehalten werden. Auch ist das Anhalten ganzer Kommandolisten nicht möglich. Wenn z.B. die Kommandoliste *a;b;c* aufgerufen wurde, und der Job *b* wird angehalten, dann fährt die **cs**h sofort mit der Ausführung von Job *c* fort. Dies ist besonders für Aliase wichtig, die ganze Kommandolisten als Werte besitzen. Das hier erwähnte Problem kann mit Kommandoklammerung (...) gelöst werden. Wenn z.B. (*a;b;c*) aufgerufen wurde, dann kann nur die ganze Subshell als ein Job angehalten werden.

## 6.18.8 Die Kommandos notify und suspend

**notify** [*job(s)*]

Das builtin-Kommando **notify** bewirkt, daß die **cs**h sofort eine Meldung ausgibt, wenn die Ausführung der angegebenen *job(s)* beendet ist. Normalerweise meldet die **cs**h die Beendigung von Hintergrund-Jobs erst vor der Ausgabe des nächsten Prompts. Werden keine *job(s)* beim Aufruf angegeben, so bezieht sich **notify** auf den aktuellen Job (zuletzt im Hintergrund gestarteter Job oder angehaltener Job). Der aktuelle Job kann auch mit dem Kommando **jobs** ermittelt werden.

**Hinweis**

Wenn die Shell-Variable **notify** gesetzt ist, wird die Beendigung jedes Hintergrund-Jobs sofort gemeldet.

Manche **cs**h-Versionen bieten noch das builtin-Kommando

**suspend**

an. **suspend** hält die aktuelle **cs**h an, und wird meist verwendet, wenn am gleichen Terminal mit dem Kommando **su** (*switch user*) auf eine andere Benutzererkennung ohne Verlassen der aktuellen **cs**h umgeschaltet wurde. Mit **suspend** kann man dann diese neue Sitzung kurzzeitig unterbrechen, um zwischenzeitlich in der aktuellen **cs**h zu arbeiten.

**6.18.9 Beispiele zur Job-Kontrolle der cs**h

Nachfolgend wird ein **cs**h-Skript vorgestellt, welches das Achtdamen-Problem löst. Beim Achtdamen-Problem gilt es, acht Damen so auf einem Schachbrett zu positionieren, daß keine der Damen eine andere bedroht. Das zugehörige **cs**h-Skript gibt alle Lösungen aus. Dabei ist anzumerken, daß solche Probleme normalerweise in einer Programmiersprache gelöst werden, da Shell-Skripts für derartige Aufgabenstellungen viel zu langsam sind. Die Lösung als **cs**h-Skript wurde aus zwei Gründen hier vorgenommen. Zum einen sollten die vielseitigen Fähigkeiten der **cs**h aufgezeigt werden, zum anderen wurde ein zeitaufwendiges **cs**h-Skript benötigt, um die Möglichkeiten der Job-Kontrolle vorzustellen.

```
% cat achtdame ↵
#!/bin/csh
achtdame - loest das Achtdame-Problem mit Backtracking
Skript sehr zeitaufwendig.

if ($#argv == 0) then
 set spalte = (0 0 0 0 0 0 0 0)
 set spaltfrei = (1 1 1 1 1 1 1 1)
 set aufwaertsfrei = (0 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
 set abwaertsfrei = (0 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
 set zeile = 0
 echo "0" >.lsg
else
 set spalte = ($argv[1])
 set spaltfrei = ($argv[2])
 set aufwaertsfrei = ($argv[3])
 set abwaertsfrei = ($argv[4])
 set zeile = $argv[5]
endif
@ zeile++
@ s = 1
while ($s <= 8)
 @ z = $zeile + $s
 @ zx = $zeile + 8 - $s
 if ($spaltfrei[$s] && $aufwaertsfrei[$z] && $abwaertsfrei[$zx])
```



```
then
 set spalte[$zeile] = $s
 set spaltfrei[$s] = 0
 set aufwaertsfrei[$z] = 0
 set abwaertsfrei[$zx] = 0
 if ($zeile == 8) then
 set lsg = `cat .lsg`
 @ lsg++
 echo "$lsg" >.lsg
 echo "${lsg}.Loesung:"
 echo " "
 @ i = 1
 while ($i <= 8)
 echo "|---|---|---|---|---|---|---|---|"
 @ j = 1
 while ($j <= 8)
 if ($spalte[$j] == $i) then
 echo "| X \c"
 else
 echo "| \c"
 endif
 @ j++
 end
 echo "|"
 @ i++
 end
 echo "|---|---|---|---|---|---|---|---|"
 echo "\n\nWeiter mit Return-Taste"
 set xx = `line` # oder: set xx = $<
 else
 # Rekursiver Aufruf des Skripts
 # Option -f legt fest: .cshrc nicht lesen
 csh -f $0 "$spalte" \
 "$spaltfrei" \
 "$aufwaertsfrei" \
 "$abwaertsfrei" \
 "$zeile"

 endif
 set spaltfrei[$s] = 1
 set aufwaertsfrei[$z] = 1
 set abwaertsfrei[$zx] = 1
endif
@ s++
end
@ zeile--

exit 0
% chmod u+x achtdame
% achtdame
[Strg-z]
Stopped [Meldung, dass achtdame angehalten]
```

```
% jobs -l
+ [1] 1329 Stopped achtdame
% stty tostop [Hintergrund-Jobs bei Ausgabe anhalten]
% bg %acht
[1] achtdame
% ls c* [Weiterarbeiten im Vordergrund]
copy
+ [1] Stopped (tty output) achtdame
% fg %1
achtdame [Meldung, dass achtdame wieder ausgeführt wird]
```

1. Loesung:

```
|---|---|---|---|---|---|---|
X						
						X
			X			
						X
	X					
			X			
					X	
		X				
```

Weiter mit Return-Taste .....

```

[Strg-z]
Stopped [Meldung, dass achtdame angehalten]
% bg %?dame
[1] achtdame
% ls a* [Weiterarbeiten im Vordergrund]
achtdame
add
add.c
ausgab
+ [1] Stopped (tty output) achtdame
% fg
achtdame [Meldung, dass achtdame wieder ausgeführt wird]
```

2. Loesung:

```
|---|---|---|---|---|---|---|
X						
						X
```

```

|---|---|---|---|---|---|---|
			X			
					X	
						X
	X					
				X		
		X				

```

Weiter mit Return-Taste .....

```
↵
```

[Strg-z]

Stopped

[Meldung, dass achtdame angehalten]

```
% bg %1 ↵
```

```
[1] achtdame
```

```
% jobs -l ↵
```

```
+ [1] 1329 Running achtdame
```

```
% kill -KILL %1 ↵
```

```
[1] Terminated achtdame
```

```
%
```

## 6.19 Builtin-Kommandos der C-Shell

Die builtin-Kommandos sind Teile des Programms **csh**. Deswegen muß die **csh** keinen neuen Prozeß starten, um sie ablaufen zu lassen, was zur Folge hat, daß diese Kommandos auch »schneller starten« als die anderen (nicht builtin-) Kommandos. Wenn jedoch builtin-Kommandos als Komponenten in einer Pipe angegeben sind, werden bis auf das letzte Kommando alle anderen Kommandos der Pipe in einer Subshell ausgeführt.

Zunächst werden in diesem Kapitel noch einige nicht behandelte builtin-Kommandos der **csh** ausführlich besprochen, bevor dann am Ende eine Zusammenfassung aller builtin-Kommandos der **csh** gegeben wird.

In der offiziellen Dokumentation zur **csh** werden die Kommandos zur Ablaufsteuerung (**if**, **switch**, **while** usw.) als builtin-Kommandos bezeichnet, was hier aus Konsistenzgründen zu den zuvor vorgestellten Shells nicht getan wird.

### 6.19.1 Der Directory-Stack (**dirs**, **pushd** und **popd**)

Die **csh** bietet einen sogenannten *Directory Stack* an, in dem Pfadnamen von Directories gespeichert werden. Der oberste Pfadname dieses Directory-Stacks legt immer das Working-Directory fest.

Wird zu einem neuen Working-Directory nicht mit **cd**, sondern mit dem Kommando **pushd** gewechselt, wird das neue Working-Directory das oberste Element des Directory-Stacks und das vorherige Working-Directory wird als zweites Element im Directory-Stack festgehalten.

Der momentane Inhalt des Directory-Stacks kann mit dem Kommando **dirs** ausgegeben werden, wobei das oberste Stack-Element (Working-Directory) als erstes ausgegeben wird.

Mit dem Kommando **popd** ist es möglich ohne Angabe des Pfads zu einem früheren Working-Directory zu wechseln, das sich im Directory-Stack befindet.

Grundsätzlich gilt, daß der Inhalt des Directory-Stacks immer dann ausgegeben wird, wenn mehr als ein Eintrag darin enthalten ist, oder wenn dieser sich ändert.

Die einzelnen builtin-Kommandos haben dabei folgende Syntax:

#### **dirs [-l]**

gibt den momentanen Inhalt des Directory-Stacks (in einer Zeile) aus. Das oberste Stack-Element, welches immer das Working-Directory ist, wird dabei als erstes (links) ausgegeben. Bei der Ausgabe wird dabei für das Home-Directory die Kurzform **~** (Tilde) verwendet, wenn nicht die Option **-l** angegeben ist.

#### **pushd**

**pushd** *pfadname*

**pushd** **+n**

Wird **pushd** ohne jegliche Argumente aufgerufen, vertauscht es die beiden obersten Directory-Pfade des Stacks.

Wird ein *pfadname* angegeben, wird dieser *pfadname* das neue Working-Directory und an oberster Stelle des Stacks hinterlegt, so daß sich der Pfadname des alten Working-Directorys dann an zweitoberster Stelle des Directory-Stacks befindet.

Bei der Angabe **+n** wird der Directory-Stack so rotiert, daß das *n*.te Element (Pfadname) als oberstes Stack-Element erscheint. Die Stack-Elemente sind dabei von 0 (oberstes Stack-Element) beginnend aufsteigend durchnumeriert.

#### **popd** [**+n**]

Wenn kein Argument angegeben ist, wird der Pfad des an oberster Stelle stehenden Directorys des Directory-Stacks gelöscht. Das an zweitoberster Stelle des Directory-Stacks stehende Directory wird zum neuen Working-Directory.

Bei der Angabe  $+n$  wird der  $n$ .te Pfad aus dem Directory-Stack entfernt. Die Stack-Elemente sind dabei wieder von 0 (oberstes Stack-Element) beginnend aufsteigend durchnummeriert.

### Beispiel

```
% pwd [↵]
/user1/egon/cshueb
% dirs [↵] [Gib momentanen Inhalt des Dir.-Stacks (mit Kurzform)
aus]
~/cshueb
% dirs -l [↵] [Gib momentanen Inhalt des Dir.-Stacks (ohne
Kurzform) aus]
/user1/egon/cshueb
% pushd /usr/include [↵] [Wechsle zu /usr/include und hinterlege es im Stack]
/usr/include ~/cshueb
% pushd /bin [↵] [Wechsle zu /bin und hinterlege es im Dir.-Stack]
/bin /usr/include ~/cshueb
% pushd [↵] [Vertausche die beiden obersten Stack-Elemente]
/usr/include /bin ~/cshueb
% pwd [↵]
/usr/include
% pushd /usr/bin [↵] [Wechsle zu /usr/bin und hinterlege es im Dir.-Stack]
/usr/bin /usr/include /bin ~/cshueb
% pushd +2 [↵] [Rotiere Stack, so daß 3.Element oberstes
Stack-Element wird]
/bin ~/cshueb /usr/bin /usr/include
% pwd [↵]
/bin
% popd [↵] [Entferne oberstes Stack-Element und mache 2.Element
zum work.dir]
~/cshueb /usr/bin /usr/include
% pwd [↵]
/user1/egon/cshueb
% pushd +2 [↵] [Rotiere Stack, so daß 3.Element oberstes
Stack-Element wird]
/usr/include ~/cshueb /usr/bin
% pushd +3 [↵]
pushd: Directory stack not that deep.
% pwd [↵]
/usr/include
% pushd +2 [↵] [Rotiere Stack, so daß 3.Element oberstes
Stack-Element wird]
/usr/bin /usr/include ~/cshueb
% pushd /bin [↵] [Wechsle zu /bin und hinterlege es im Dir.-Stack]
/bin /usr/bin /usr/include ~/cshueb
% popd [↵] [Entferne oberstes Stack-Element und mache
2.Element zum work.dir]
/usr/bin /usr/include ~/cshueb
% popd +1 [↵] [Lösche 2.Element aus Dir.-Stack]
/usr/bin ~/cshueb
% popd [↵] [Entferne oberstes Stack-Element und mache
2.Element zum work.dir]
```

```
~/cshueb
% pwd
/user1/egon/cshueb
%
```

**Hinweise**

Auf manchen **csh**-Versionen sind die builtin-Kommandos **pushd** und **popd** als Aliase realisiert.

Manche **csh**-Versionen bieten zusätzlich noch die beiden Aliase (builtin-Kommandos) **swapd** und **flipd** an:

**swapd**

vertauscht die beiden Directory-Pfade, die an zweit- und dritt-oberster Stelle des Stacks stehen.

**flipd**

entspricht dem Aufruf von **pushd** ohne Argumente, der ein Vertauschen der beiden obersten Pfade des Directory-Stacks durchführt. Es wird also ständig zwischen diesen beiden Directories hin und her gewechselt.

## 6.19.2 Ausgeben von Text (echo und glob)

Die **csh** kennt zwei builtin-Kommandos, um Text auszugeben:

**echo** [-n] [*argument(e)*]

**echo** gibt die *argument(e)*, nachdem für diese Parametersubstitution, Kommandosubstitution und Dateinamen-Expandierung durchgeführt wurde, auf die Standardausgabe aus. Bei der Ausgabe werden alle Argumente durch ein Leerzeichen voneinander getrennt. Diese gesamte Ausgabe wird dann mit einem Neuezeile-Zeichen abgeschlossen.

Ist die Option **-n** angegeben, wird die Ausgabe nicht mit einem Neuezeile-Zeichen abgeschlossen.

Auf manchen **csh**-Versionen läßt das **echo**-Kommando folgende C-ähnliche Notationen zu:

**\c** gibt die angegebenen *argument(e)* bis zu diesem Zeichen aus, und unterdrückt den üblichen Zeilenvorschub durch das Kommando **echo**. Eventuell danach angegebene Argumente werden nicht ausgegeben.

**\n** Neuezeile-Zeichen

**glob** [*argument(e)*]

entspricht weitgehend dem Kommando **echo**, nur daß die oben angegebenen Escape-Sequenzen ihre Sonderbedeutung verlieren und die einzelnen *argument(e)* bei der Ausgabe nicht durch Leerzeichen getrennt, sondern wie in C mit **\0** abgeschlossen werden; auch wird die Ausgabe nicht mit einem Neuezeile-

Zeichen abgeschlossen. **glob** wird oft in C-Programmen benutzt, um eine Dateinamen-Expanding durch die **csh** vornehmen zu lassen.

**Beispiel**

```
% pwd ↵
/user1/egon/cshueb
% echo a* ↵
achtdame add add.c ausgab
% glob a* ↵
achtdameaddadd.causgab% ↵
% glob a* | od -cb ↵
0000000 a c h t d a m e \0 a d d \0 a d d
 141 143 150 164 144 141 155 145 000 141 144 144 000 141 144 144
0000020 . c \0 a u s g a b \0
 056 143 000 141 165 163 147 141 142 000
0000031
%
```

### 6.19.3 Der Hashing-Mechanismus (hashstat, rehash und unhash)

Die **csh** arbeitet eine eingegebene Kommandozeile in folgender Reihenfolge (siehe auch Kapitel 6.20) ab:

1. History-Substitution
2. Alias-Substitution
3. Parameter-Substitution
4. Kommando-Substitution und Dateinamen-Expanding

Das erste Wort legt nach diesen Ersetzungen dann den Namen des aufzurufenden Programms fest. Handelt es sich dabei um ein Kommando zur Ablaufsteuerung (**if**, **while** usw.) oder um ein builtin-Kommando, führt die **csh** den betreffenden Programmteil aus, andernfalls durchsucht sie ihre interne Hashing-Tabelle nach dem als erstes Wort angegebenen Programmnamen (Kommandonamen).

Um ein ständiges zeitaufwendiges Suchen in den **path**-Directories beim Aufruf eines nicht builtin-Kommandos zu vermeiden, unterhält die **csh** ähnlich zur Bourne-Shell eine interne Hashing-Tabelle, in der sie die Pfadnamen aller Kommandos der **path**-Directories speichert. Im Unterschied zur Bourne-Shell wird diese Hashing-Tabelle jedoch nicht bei jedem Aufruf eines Kommandos ständig erweitert, sondern bereits am Beginn der UNIX-Sitzung vollständig erstellt.

Die **cs**h bietet nun drei builtin-Kommandos an, um Informationen aus der Hashing-Tabelle abzufragen bzw. diese zu manipulieren:

### hashstat<sup>1</sup>

gibt eine Statistik aus, die über den Erfolgsgrad der internen Hashing-Tabelle informiert. Diese Statistik zeigt an, wie viele aufgerufene Kommandos auch wirklich gefunden wurden und wie viele nicht; dazu wird noch eine Prozentzahl angegeben.

### rehash

bewirkt, daß die interne Hashing-Tabelle zunächst gelöscht und dann wieder neu aufgebaut wird. **rehash** muß immer dann aufgerufen werden, wenn während einer UNIX-Sitzung neue Programme in den **path**-Directories angelegt wurden. Da in der **cs**h anders als in der Bourne-Shell die Hashing-Tabelle bereits am Anfang der Sitzung aufgebaut wurde, würden diese Programme niemals in der veralteten Hashing-Tabelle gefunden, außer sie werden mit dem absoluten Pfadnamen aufgerufen.

### unhash

schaltet die Verwendung der Hashing-Tabelle zum Lokalisieren von Kommandos aus.

#### Beispiel

```
% pwd ↵
/user1/egon/cshueb
% echo $path ↵
/bin /usr/bin .
% fiba ↵ [fiba im work. dir. wird gefunden]
1
1
2
:
:
610
987
% set path = (/bin /usr/bin $home .) ↵
% mv fiba .. ↵ [Verlagere fiba nach parent dir. (home dir.)]
% fiba ↵ [Nun wird fiba nicht gefunden, obwohl $home]
fiba: Command not found. [in path-Directories vorhanden]
% rehash ↵ [Baue Hashing-Tabelle neu auf]
% fiba ↵ [Nun wird fiba wieder gefunden]
1
1
2
:
:
610
```

---

1. Nicht auf allen **cs**h-Versionen verfügbar.



```
987
% pwd
/user1/egon/cshueb
% mv ../fiba .
% [fiba wieder nach /user1/egon/cshueb
% verlagern]
```

**Hinweis**

Wenn das Working-Directory in der Shell-Variablen **path** vorkommt, sucht die **csh** immer im Working-Directory und verläßt sich dabei nicht auf die Hashing-Tabelle. So würde z.B. ein erneuter Aufruf von **fiba** im obigen Beispiel **fiba** auffinden, auch wenn die Hashing-Tabelle nicht mit **rehash** aktualisiert wurde.

## 6.19.4 Setzen und Löschen von Environment-Variablen (setenv und unsetenv)

Shell-Variablen, die mit **set** definiert sind, werden normalerweise nicht in die Environment von Kindprozessen exportiert. Die Ausnahmen dabei sind die Shell-Variablen **term** und **path**. Um ein Exportieren von Shell-Variablen zu erreichen, müssen diese mit dem builtin-Kommando **setenv**, welches in etwa dem **export** der Korn-Shell entspricht, definiert werden. So definierte Shell-Variablen können mit **unsetenv** auch wieder aus der Environment gelöscht werden:

**setenv** *variablenname wert*

definiert eine Environment-Variable *variablenname* mit dem angegebenen *wert*.

**unsetenv** *pattern*

löscht alle Environment-Variablen, welche durch das angegebene *pattern* abgedeckt werden. Im *pattern* sind dabei die Metazeichen \*, ? und [...] der Dateinamen-Expansion erlaubt. Es ist kein Fehler, wenn die angegebenen *pattern* keinen einzigen Namen einer Environment-Variablen abdecken.

**Hinweis**

Um sich alle momentan definierten Environment-Variablen anzeigen zu lassen, steht das nicht-builtin-Kommando **env** bzw. **printenv** zur Verfügung.

**Beispiel**

```
% set x = 100
% echo $x
100
% csh
% echo $x
x: Undefined variable.
% exit
% setenv y 100
% echo $y
100
% csh
% echo $y
100
% env
[Nun ist y in Subshell bekannt]
```

```
HOME=/user1/egon
PATH=/bin:/usr/bin:/user1/egon:
LOGNAME=egon
TERM=vt100
SHELL=/bin/csh
MAIL=/usr/mail/egon
y=100
% exit
% set path = (/bin /usr/bin .)
% env [Das Ändern von path hat sich auch auf die Env.-Var. PATH
 ausgewirkt]

HOME=/user1/egon
PATH=/bin:/usr/bin:
LOGNAME=egon
TERM=vt100
SHELL=/bin/csh
MAIL=/usr/mail/egon
y=100
%
```

### 6.19.5 Werte von Arrays verschieben (shift)

Um die Positionsparameter nach links zu verschieben, steht das builtin-Kommando **shift** zur Verfügung:

**shift** [*variable*]

Wird **shift** ohne Argument aufgerufen, werden die Elemente von **argv** um eine Position nach links geschoben, so daß der Wert **\$argv[2]** dem Positionsparameter **argv[1]**, der Wert **\$argv[3]** dem Positionsparameter **argv[2]** usw. zugewiesen wird.

Wird eine *variable* angegeben, werden die einzelnen Elemente dieser *variable* um eine Position nach links geschoben.

Wenn **argv** bzw. die angegebene *variable* keinen Wert mehr besitzt, wertet **shift** dies als Fehler.

**Beispiel**

```
% set argv = (eins zwei drei vier fuenf)
% echo $argv
eins zwei drei vier fuenf
% echo $#argv
5
% shift
% echo $argv
zwei drei vier fuenf
% shift
% echo $argv
drei vier fuenf
% echo $*
drei vier fuenf
```

```
% echo $#argv ↵
3
% set zahl = (one two three four five six) ↵
% echo $#zahl ↵
6
% shift zahl ↵
% echo $zahl ↵
two three four five six
% echo $#zahl ↵
5
% shift zahl ↵
% echo $zahl ↵
three four five six
% echo $#zahl ↵
4
% set y = "" ↵
% echo $#y ↵
1
% shift y ↵
% echo $#y ↵
0
% shift y ↵
shift: No more words.
% echo $#y ↵
0
% echo $?y ↵
1
% [y ist zwar leer, aber noch definiert]
%
```

### 6.19.6 Das Punkt-Kommando der csh (source)

**source** ist das Gegenstück der **csh** zum Punkt-Kommando der Bourne-Shell:

**source** *dateiname*

**source** liest die Kommandos der angegebenen Datei *dateiname* und führt sie in der aktuellen **csh** aus. **source** wird häufig verwendet, wenn der Inhalt der Dateien *.login* und *.cshrc* verändert wurde. Um diese Änderungen der aktuellen **csh** bekannt zu machen, ohne daß sich der betreffende Benutzer ab- und wieder anmeldet, kann

**source** *\$home/.login* bzw.

**source** *\$home/.cshrc*

aufgerufen werden.

**Beispiel**

```
% echo $name ↵
name: Undefined variable.
% echo set name = Emil >namdat ↵
% source namdat ↵
% echo $name ↵
Emil
%
```

**Hinweise** In mit **source** gelesenen Dateien können wieder **source**-Aufrufe stehen, so daß ein geschachteltes Lesen von Kommandodateien möglich ist.

Die mit **source** gelesenen Kommandos werden nicht im History-Puffer gespeichert.

### 6.19.7 Priorität von Kommandos herabsetzen (nice)

Mit dem builtin-Kommando **nice** ist es möglich, die Priorität von Kommandos bei ihrer Ausführung herabzusetzen, so daß sie langsamer ablaufen, da sie sich bei der Verteilung der CPU-Zeit freiwillig hinten anstellen, was natürlich für die anderen Prozesse, die ebenfalls um CPU-Zeit konkurrieren, von Vorteil ist:

**nice** [+n] [*kommando*]

Wird **nice** ohne Argumente aufgerufen, so wird der *nice*-Wert auf 4 gesetzt.

Wird nur +n angegeben, wird der *nice*-Wert auf die Zahl n gesetzt.

Wird **nice** mit einem *kommando* aufgerufen, wird nur dieses *kommando* mit *nice*-Wert 4 bzw. *nice*-Wert n, wenn +n angegeben ist, ausgeführt.

**Hinweis** Je größer der *nice*-Wert ist, um so niedriger ist die Priorität, daß die entsprechenden Kommandos um so langsamer ausgeführt werden.

Nur der Superuser kann Kommandos mit einem negativen *nice*-Wert (-n) ausführen lassen.

### 6.19.8 Zeitmessungen für Kommandos (time)

**time** gibt aus, wieviel Zeit von der **cs**h bzw. von einem einzelnen Kommando verbraucht wurde:

**time** [*kommando*]

Wird **time** ohne Argument aufgerufen, gibt es die von der aktuellen **cs**h und allen ihren Kindprozessen bis zum jetzigen Zeitpunkt verbrauchte Zeit aus.

Wird ein *kommando* angegeben, gibt **time** nach der Ausführung des Kommandos die von diesem *kommando* verbrauchte Zeit aus.

Die verbrauchte Zeit wird dabei durch drei Zeitwerte und eine Prozentzahl angezeigt:

*user sys elapsed prozentzahl*

Die Prozentzahl drückt dabei das Verhältnis von  $(user+sys) : elapsed$  aus.

**Beispiel**

```
% time ↵
0.6u 2.5s 28:16 0%
% time fiba ↵
1
1
2
3
:
:
610
987
0.9u 0.4s 0.01 126%
$ time sleep 10 ↵
0.0u 0.1s 0:10 1%
%
```

## 6.19.9 An- und Abmelden mit login und logout

Diese beiden builtin-Kommandos haben folgende Syntax:

**login** [*loginname*]<sup>1</sup>

beendet die Login-Shell und startet die Login-Prozedur für eine neue UNIX-Sitzung; wird ein *loginname* angegeben, wird sofort nach dem Paßwort des entsprechenden Benutzers gefragt.

**logout**

beendet eine Login-Shell, wobei zuvor noch die Datei *.logout* (im Home-Directory) ausgeführt wird.

**Hinweis**

Wenn die Shell-Variable **ignoreeof** gesetzt ist, kann eine interaktive **cs**h nur mit den builtin-Kommandos **exit** und **logout**, aber nicht mit *Strg-D* beendet werden.

---

1. Nicht auf allen **cs**h-Versionen verfügbar.

## 6.19.10 Limits für Systemressourcen (limit und unlimited)

Mit den builtin-Kommandos **limit** und **unlimit** können Obergrenzen für Systemressourcen (wie z.B. CPU-Zeit, Dateigröße usw.) festgelegt und auch wieder aufgehoben werden:

**limit** [*resource*] [*maximum*]<sup>1</sup>

legt ein maximales Limit (*maximum*) für die angegebene *resource* fest. Dieses Limit darf vom aktuellen Prozeß und allen seinen Kindprozessen nicht überschritten werden.

Ist kein *maximum* angegeben, wird momentane Limit für *resource* oder für alle Ressourcen ausgegeben, wenn *resource* nicht angegeben ist.

Für *resource* kann dabei folgendes angegeben werden:

| <i>resource</i>      | <i>Bedeutung</i>                 |
|----------------------|----------------------------------|
| <b>cpulimit</b>      | CPU-Zeit in Sekunden             |
| <b>filesize</b>      | Dateigröße in Bytes              |
| <b>datasize</b>      | Größe des Datensegments in Bytes |
| <b>stacksize</b>     | Größe des Stacksegments in Bytes |
| <b>coredump-size</b> | <i>coredump</i> -Größe in Bytes  |

Für *maximum* kann eine Gleitpunkt-Zahl oder eine ganze Zahl angegeben werden.

Am Ende dieser Zahl kann bei **filesize**, **datasize**, **stacksize** und **coredumpsize** einer der folgenden Buchstaben angegeben werden:

**k** Zahl gibt Kilobytes an

**m** Zahl gibt Megabytes an

Bei **cpulimit** kann am Ende dieser Zahl einer der folgenden Buchstaben angegeben werden:

**m** Zahl gibt Minuten an

**h** Zahl gibt Stunden an

Außerdem kann bei **cpulimit** statt einer Zahl auch *mm:ss* angegeben werden, wobei *mm* die Minuten- und *ss* die Sekundenanzahl festlegt.

1. Nicht auf allen **cs**-Versionen verfügbar.

### **unlimit** [*resource*]<sup>1</sup>

hebt das für *resource* festgelegte Limit wieder auf. Ist *resource* nicht angegeben, werden die Limits für alle System-Ressourcen entfernt.

#### **Beispiel**

```
% limit ↵
cputime unlimited
filesize unlimited
datasize 44964 kbytes
stacksize 22482 kbytes
coredumpsize unlimited
% limit coredumpsize 512k ↵
% limit ↵
cputime unlimited
filesize unlimited
datasize 44964 kbytes
stacksize 22482 kbytes
coredumpsize 512k
%
```

## 6.19.11 Zusammenfassung der builtin-Kommandos

Hier werden alle builtin-Kommandos der **csh** nochmals zusammengefaßt. Dabei wird eine kurze Beschreibung gegeben.<sup>2</sup>

Die **csh** verfügt über folgende builtin-Kommandos:

| Kommando                                          | Wirkung                                                                      |
|---------------------------------------------------|------------------------------------------------------------------------------|
| <b>#</b>                                          | Kommentar-Kommando                                                           |
| <b>alias</b> [ <i>name</i> ] [ <i>wortliste</i> ] | Definieren bzw. Anzeigen von Aliase                                          |
| <b>bg</b> [ <i>job(s)</i> ]                       | Ausführung von angehaltenen <i>job(s)</i> im Hintergrund fortsetzen          |
| <b>break</b>                                      | Verlassen einer umschließenden <b>while</b> - oder <b>foreach</b> -Schleife. |
| <b>cd</b> [ <i>directory</i> ]                    |                                                                              |
| <b>chdir</b> [ <i>directory</i> ]                 | Wechseln in ein anderes Working-Directory                                    |
| <b>continue</b>                                   | Abbrechen eines <b>while</b> - oder <b>foreach</b> -Schleifendurchlaufs      |
| <b>dirs</b> [-l]                                  | Ausgeben des Directory-Stacks                                                |
| <b>echo</b> [-n] [ <i>argument(e)</i> ]           | Ausgeben der <i>argument(e)</i> auf die Standardausgabe                      |
| <b>eval</b> <i>argument(e)</i>                    | Ausführen der <i>argument(e)</i> als Kommandos                               |

1. Nicht auf allen **csh**-Versionen verfügbar.

2. Eine genauere Beschreibung der jeweiligen builtin-Kommandos findet sich in »Linux-Unix-Kurzreferenz«

| Kommando                                            | Wirkung                                                                                                            |
|-----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| <b>exec</b> <i>argument(e)</i>                      | Überlagern der <b>cs</b> h mit einem Kommando                                                                      |
| <b>exit</b> [( <i>ausdr</i> )]                      | Beenden der momentan aktiven <b>cs</b> h                                                                           |
| <b>fg</b> [ <i>job(s)</i> ]                         | Ausführung von angehaltenen Jobs oder Hintergrund-Jobs im Vordergrund fortsetzen                                   |
| <b>glob</b> [ <i>argument(e)</i> ]                  | Ausgeben der <i>argument(e)</i> auf die Standardausgabe                                                            |
| <b>hashstat</b>                                     | Ausgeben einer Statistik, die über den Erfolgsgrad der internen Hashing-Tabelle informiert.                        |
| <b>history</b> [-r] [ <i>n</i> ]                    | Anzeigen der Kommandos im History-Puffer                                                                           |
| <b>jobs</b> [-l]                                    | Ausgeben von Informationen zu angehaltenen Jobs und Hintergrund-Jobs                                               |
| <b>kill</b> [- <i>signal</i> ] <i>job(s)</i>        | Signale an Jobs schicken                                                                                           |
| <b>kill -l</b>                                      | Verfügbare Signale auflisten                                                                                       |
| <b>limit</b> [ <i>resource</i> ] [ <i>maximum</i> ] | Festlegen bzw. Anzeigen von Limits für Systemressourcen                                                            |
| <b>login</b> [ <i>loginname</i> ]                   | Starten der Login-Prozedur                                                                                         |
| <b>logout</b>                                       | Verlassen einer Login-Shell                                                                                        |
| <b>newgrp</b> [ <i>gruppenname</i> ]                | Wechseln der Gruppenzugehörigkeit                                                                                  |
| <b>nice</b> [+ <i>n</i> ] [ <i>kommando</i> ]       | Kommandos mit niedrigerer Priorität ausführen lassen                                                               |
| <b>nohup</b> [ <i>kommando</i> ]                    | Ignorieren von <i>nohup</i> -Signalen                                                                              |
| <b>notify</b> [ <i>job(s)</i> ]                     | Sofortiges Informieren bei Beendigung von Hintergrund-Jobs                                                         |
| <b>onintr</b> [-] [ <i>marke</i> ]                  | Abfangen von Signalen                                                                                              |
| <b>popd</b> [+ <i>n</i> ]                           | Wechseln zu einem Directory des Directory-Stacks bzw. Entfernen eines Directories aus dem Directory-Stack          |
| <b>pushd</b> [ <i>pfadname</i>   + <i>n</i> ]       | Hinzufügen eines Directories zum Directory-Stack bzw. Vertauschen oder Rotieren der Directories im Directory-Stack |
| <b>rehash</b>                                       | Interne Hashing-Tabelle neu erstellen lassen                                                                       |
| <b>set</b>                                          | Setzen / Anzeigen von Variablen bzw. Zuweisen von Werten / Wortlisten an Variablen:                                |
| <b>set</b>                                          | gibt alle momentan definierten Shell-Variablen mit ihren Werten aus                                                |
| <b>set</b> <i>variablenname</i>                     | weist der Variablen <i>variablenname</i> den Nullstring zu                                                         |
| <b>set</b> <i>variablenname</i> = <i>wert</i>       | weist der Variablen <i>variablenname</i> den Wert <i>wert</i> zu                                                   |



| Kommando                                                          | Wirkung                                                                                                                                                                                                                                                                                    |
|-------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>set</b> <i>variablenname</i> =(wortliste)                      | weist dem Array <i>variablenname</i> die Werte von <i>wortliste</i> zu; dabei wird der erste Wert <i>variablenname</i> [1], der zweite <i>variablenname</i> [2] usw. zugewiesen                                                                                                            |
| <b>set</b> <i>variabl#enname</i> [ <i>index</i> ]=wert            | weist dem Array-Element <i>variablenname</i> [ <i>index</i> ] den Wert <i>wert</i> zu; dabei muß <i>variablenname</i> [ <i>index</i> ] bereits existieren.                                                                                                                                 |
| <b>setenv</b> <i>variablenname</i> wert                           | Definieren von Environment-Variablen                                                                                                                                                                                                                                                       |
| <b>shift</b> [ <i>variable</i> ]                                  | Verschieben der Werte von Positionsparametern oder Variablen                                                                                                                                                                                                                               |
| <b>source</b> <i>dateiname</i>                                    | Lesen und Ausführen der Kommandos aus <i>dateiname</i> in der aktuellen <b>csh</b>                                                                                                                                                                                                         |
| <b>stop</b> [ <i>job(s)</i> ]                                     | Anhalten von Hintergrund- <i>job(s)</i>                                                                                                                                                                                                                                                    |
| <b>suspend</b>                                                    | Anhalten der aktuellen <b>csh</b>                                                                                                                                                                                                                                                          |
| <b>time</b> [ <i>kommando</i> ]                                   | Ausgeben der von der aktuellen <b>csh</b> bzw. von einem Kommando gebrauchten CPU-Zeit                                                                                                                                                                                                     |
| <b>umask</b> [ <i>3-stellige-oktalzahl</i> ]                      | Setzen bzw. Anzeigen der Dateikreierungs-Maske                                                                                                                                                                                                                                             |
| <b>unalias</b> <i>pattern</i> ...                                 | Löschen von Aliase                                                                                                                                                                                                                                                                         |
| <b>unhash</b>                                                     | Verwendung der internen Hashing-Tabelle durch die <b>csh</b> ausschalten                                                                                                                                                                                                                   |
| <b>unlimit</b> [ <i>resource</i> ]                                | Beseitigen von Limits für Systemressourcen                                                                                                                                                                                                                                                 |
| <b>unset</b> <i>pattern</i> ...                                   | Löschen von Shell-Variablen                                                                                                                                                                                                                                                                |
| <b>unsetenv</b> <i>pattern</i> ...                                | Löschen von Environment-Variablen                                                                                                                                                                                                                                                          |
| <b>wait</b>                                                       | Warten auf die Beendigung von Hintergrund-Jobs                                                                                                                                                                                                                                             |
| @                                                                 | Auswerten von arithmetischen Ausdrücken:                                                                                                                                                                                                                                                   |
| @                                                                 | Ausgeben der Werte aller momentan definierten Shellvariablen                                                                                                                                                                                                                               |
| @ <i>variable</i> = <i>ausdruck</i> <sup>a</sup>                  | setzt die <i>variable</i> mit dem Ergebnis, das die Auswertung von <i>ausdruck</i> liefert                                                                                                                                                                                                 |
| @ <i>variable</i> [ <i>index</i> ] = <i>ausdruck</i> <sup>b</sup> | setzt das Array-Element <i>variable</i> [ <i>index</i> ] mit dem Ergebnis, das die Auswertung von <i>ausdruck</i> liefert. Hierbei ist zu beachten, daß <i>variable</i> [ <i>index</i> ] bereits existieren muß, also bereits zuvor mit <b>set</b> oder <b>setenv</b> deklariert sein muß. |

a. Die Leerzeichen vor und nach dem Zuweisungsoperator = sind nicht notwendig.

b. Die Leerzeichen vor und nach dem Zuweisungsoperator = sind nicht notwendig.

## 6.20 Die Abarbeitung von Kommandozeilen

Die Abarbeitung von Kommandozeilen wird in folgender Reihenfolge vorgenommen:

1. Entfernen von allen \Neuezeile-Zeichen
2. History-Substitution
3. Speichern der Kommandozeile im History-Puffer
4. Alias-Substitution
5. Parameter-Substitution
6. Kommando-Substitution
7. Dateinamen-Expandierung (einschließlich Tilde-Expandierung)
8. Ein- / Ausgabeumlenkungen
9. Kommandoausführung

**Beispiel**

```
% set history = 10
% alias h history
% alias gibaus drucke
% alias drucke echo
% set b = 'b*'
% set echo
% h
history
1 set history = 10
2 alias h history
3 alias gibaus drucke
4 alias drucke echo
5 set b = 'b*'
6 set echo
7 h
% !3:1 a* \
$b `echo c*` ~
gibaus a* $b `echo c*` ~
echo a* b* `echo c*` ~
echo c*
achtdame add add.c ausgab copy
% h
history
1 set history = 10
2 alias h history
3 alias gibaus drucke
4 alias drucke echo
5 set b = 'b*'
6 set echo
7 h
```

```
8 gibus a* $b `echo c` ~
9 h
%
```

Bei der Kommandoausführung wird zunächst geprüft, ob es sich beim ersten Wort um ein builtin-Kommando handelt. Wenn dies zutrifft, wird der entsprechende Programmteil der **cs**h ausgeführt.

Handelt es sich um kein builtin-Kommando, verwendet die **cs**h, wenn die Optionen **-c** oder **-t** nicht gesetzt sind, ihre interne Hashing-Tabelle, um den absoluten Pfadnamen zu diesem Kommando zu bestimmen. Falls in der Shell-Variablen **path** Directory-Pfade angegeben sind, die nicht mit / beginnen, umgeht die **cs**h dafür den Hashing-Mechanismus und sucht in diesen Directories nach dem Kommandonamen.

Falls mit **unhash** dieser interne Hashing-Mechanismus ausgeschaltet ist oder die Optionen **-c** oder **-t** gesetzt sind, durchsucht die **cs**h die in **path** angegebenen Directories nach dem Kommandonamen.

## 6.21 Aufrufsyntax der cs

Die mögliche Aufrufsyntax für die **cs**h ist:

**cs** [-cefinstvVxX] [*argument(e)*]

Die Optionen bedeuten dabei folgendes:

| Optionen       | Wirkung                                                                                                                                                                                                  |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-c text</b> | ( <b>command</b> ) <i>text</i> legt dabei die auszuführenden Kommandos fest. Sind weitere Argumente angegeben, so werden diese in <b>argv</b> hinterlegt.                                                |
| <b>-e</b>      | ( <b>exit</b> ) <b>cs</b> h verlassen, wenn ein Kommando nicht erfolgreich ausgeführt werden kann.                                                                                                       |
| <b>-f</b>      | Datei <b>\$home/.cshrc</b> beim Start der <b>cs</b> h nicht ausführen; dies bewirkt, daß die <b>cs</b> h schneller startet.                                                                              |
| <b>-i</b>      | ( <b>interactive</b> ) als interaktive <b>cs</b> h starten; eine <b>cs</b> h wird immer automatisch als interaktive <b>cs</b> h gestartet, wenn die Ein- und Ausgaben auf das Terminal eingestellt sind. |
| <b>-n</b>      | ( <b>noexec</b> ) Kommandos werden nur gelesen und auf Syntaxfehler untersucht, aber nicht ausgeführt. Diese Option kann für eine syntaktische Überprüfung von <b>cs</b> h-Skripts verwendet werden.     |

| Optionen                            | Wirkung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-s</b><br>[ <i>argument(e)</i> ] | <p>bewirkt, daß eine interaktive Subshell gestartet wird, die Kommandos von der Standardeingabe liest. Wenn <i>argument(e)</i> angegeben sind, dann werden diese der Subshell als Positionsparameter übergeben. Diese Option wird automatisch eingeschaltet, wenn die Option <b>-c</b> nicht angegeben ist. Falls die Option <b>-c</b> angegeben ist, so wird die Option <b>-s</b> ignoriert.</p> <p>Beispiel:</p> <pre>% pwd ↵ /user1/egon/cshueb % csh -s egon `pwd` ↵ % echo "Hallo, \$1" ↵ Hallo, egon ↵ % echo \$argv[2] ↵ /user1/egon/cshueb % exit ↵ %</pre> |
| <b>-t</b>                           | Nur eine Kommandozeile lesen und ausführen; \ kann verwendet werden, um die Sonderbedeutung des Neuezeile-Zeichens (Kommando ausführen) auszuschalten.                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>-v</b>                           | ( <b>verbose</b> ) bewirkt, daß die Shell-Variable <b>verbose</b> gesetzt wird, so daß nach jeder History-Substitution die neu entstandene Kommandozeile vor ihrer Ausführung angezeigt wird.                                                                                                                                                                                                                                                                                                                                                                       |
| <b>-V</b>                           | ( <b>Verbose</b> ) entspricht der Option <b>-v</b> , nur daß die Shell-Variable <b>verbose</b> bereits vor der Ausführung von <i>.cshrc</i> gesetzt wird.                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>-x</b>                           | ( <b>xtrace</b> ) bewirkt, daß die Shell-Variable <b>echo</b> gesetzt wird, so daß jede Kommandozeile vor ihrer Ausführung nochmals ausgegeben wird. Während jedoch bei nicht builtin-Kommandos alle Substitutionen und Expandierungen vor dieser Ausgabe bereits durchgeführt sind, werden bei builtin-Kommandos diese Expandierungen und Substitutionen erst nach der Ausgabe vorgenommen.                                                                                                                                                                        |
| <b>-X</b>                           | ( <b>Xtrace</b> ) entspricht der Option <b>-x</b> , nur daß die Shell-Variable <b>echo</b> bereits vor der Ausführung von <i>.cshrc</i> gesetzt wird.                                                                                                                                                                                                                                                                                                                                                                                                               |

Die meisten dieser Optionen können auch beim Aufruf eines **csh**-Skripts gesetzt werden:

**csh** [*optionen*] *skript*

In diesem Fall gelten die gesetzten Optionen nur für die Dauer der Skript-Ausführung.

Bei jedem Start einer **csh** werden die Dateien */etc/cshrc* und **\$home/.cshrc** gelesen und ausgeführt, wenn diese Dateien existieren.

Die Login-Prozedur ruft die **csh** mit

### **exec -csh**

auf. Das Minuszeichen - bewirkt, daß in diesem Fall eine Login-Shell gestartet wird. Eine Login-Shell unterscheidet sich von einer normalen Shell nur in den folgenden Punkten:

- ▶ Es können außer Optionen keine Argumente angegeben werden.
- ▶ Nach dem Lesen und Ausführen der Kommandos in den beiden Dateien */etc/cshrc* und *\$home/.cshrc* werden als nächstes die Kommandos aus der Datei *\$home/.login* gelesen und ausgeführt, wenn diese Datei existiert.

## 6.22 Einrichten einer persönlichen Arbeitsumgebung

Da die **csh** nicht wie die Bourne-Shell das Kommando **export** anbietet, um mit **set** definierte Variablen an Subshells zu exportieren, muß sie eine Möglichkeit anbieten, um auch eine Environment für eine Subshell aufzubauen. Dies erfolgt in drei Schritten:

1. Gewisse Shell-Variablen werden aus der globalen Environment zur Verfügung gestellt. Dies sind meist die Shell-Variablen **home** und **shell**.
2. Die Variable **path** wird mit denen am jeweiligen System voreingestellten Directories gesetzt.
3. Bei jedem Aufruf einer neuen **csh** werden zuerst die Kommandos aus der vom Systemverwalter geschriebenen Datei */etc/cshrc* und dann die Kommandos aus der Datei *.cshrc* im Home-Directory des speziellen Benutzers ausgeführt.

Handelt es sich beim Aufruf einer **csh** um eine Login-Shell, werden zusätzlich noch die Kommandos aus der Datei *\$home/.login* gelesen und ausgeführt, wenn diese Datei existiert.

Beim Abmelden einer Login-Shell werden vor dem endgültigen Verlassen der **csh** noch die Kommandos aus der Datei *\$home/.logout* gelesen und ausgeführt, wenn diese Datei existiert.

Nachfolgend wird anhand von Beispielen gezeigt, wie diese Dateien aussehen können. Bis auf die Datei */etc/cshrc*, die vom Systemverwalter erstellt wird, handelt es sich nur um Vorschläge, die jeder Benutzer seinen eigenen Bedürfnissen anpassen kann.

### 6.22.1 Die Datei /etc/cshrc

Der Inhalt der Datei */etc/cshrc* kann Informationen über Maschinen-Eigenschaften und über Voreinstellungen für alle Benutzer geben. Ein Beispiel für das Aussehen einer solchen Datei */etc/cshrc* könnte sein:

```
% cat /etc/cshrc ↵
#
#
Kommentare
#
#
umask 022 # Setzen der Datei-Kreierungsmaske

Gib Message Of The Day aus
echo "" # Zeilenvorschub
if (-r /etc/motd) cat /etc/motd

Allgemeine Terminaleinstellungen
stty erase '^h' echoe

MAIL-Variable setzen und Briefkasten pruefen
if ($?MAIL == 0) setenv MAIL /usr/mail/$LOGNAME
if ((-z $MAIL || -e $MAIL) && \
 !(-z $MAIL && -e $MAIL)) \
 echo "\nYou have mail"

Neuigkeiten vom schwarzen Brett ausgeben
if ($LOGNAME != root && -x /usr/bin/news) news -n
%
```

### 6.22.2 Die Datei .cshrc

Da die Datei **\$home/.cshrc** bei jedem Start einer neuen **csh** (außer bei Option **-f**) automatisch gelesen und ausgeführt wird, sollte diese Datei nicht zu viele Kommandos enthalten. Dies wirkt sich sonst negativ auf die Startzeit von **csh**-Skripten aus. Deswegen ist es auch empfehlenswert, nur builtin-Kommandos in *.cshrc* aufzurufen, damit während der Ausführung von *.cshrc* nicht neue Prozesse kreiert werden.

Üblicherweise werden in **\$home/.cshrc** nur Aliase und Shell-Variablen wie z.B. **history**, **noclobber** usw. definiert:

```
% cat ~/.cshrc ↵
#
Beispiel fuer das Aussehen der Datei $home/.cshrc
#

Nuetzliche Alias-Definitionen
```

```
alias h history
alias rm rm -i
alias ll ls -CF

Prompt setzen und Alias cd definieren
set prompt="! `pwd`>> "
alias cd 'chdir \!*; set prompt="! `pwd`>> "'

Vordefinierte Shellvariablen setzen
set path = (/bin /usr/bin $home/bin /etc .)
set term = ansi
set cdpath = (. .. $home ~/cshueb ~/bin)
set time=20
set history=20
set mailcheck=0
set ignoreeof

tput init
%
```

### 6.22.3 Die Datei .login

Beim Start einer Login-Shell werden nach den Dateien */etc/cshrc* und **\$home/.cshrc** die Kommandos aus **\$home/.login** gelesen und ausgeführt.

Typische Aktionen in *.login* sind z.B.:

- ▶ Durchführen von einmaligen Terminal-Einstellungen mit **stty**
- ▶ Konfigurieren des **vi** über die Variable **EXINIT**
- ▶ Setzen von nützlichen globalen Variablen mit **setenv**
- ▶ Festlegen von Aktionen, die immer am Anfang einer UNIX-Sitzung durchzuführen sind, wie z.B. Ausgabe des letzten Anmeldezeitpunkts, Ausgabe des heutigen Datums mit Uhrzeit usw.
- ▶ Abgeben von **cron**- und **at**-Aufträgen, wie z.B. Einbau eines automatischen Erinnerungsservice.

Ein Beispiel für das mögliche Aussehen von *.login* wäre:

```
% cat ~/.login 
#
Beispiel fuer das Aussehen der Datei $home/.login
#

Definition von nuetzlichen Environment-Variablen
setenv hu $home/cshueb
setenv hb $home/bin

Definition der mail-Variablen
```

```
set mail=/usr/mail/`logname`
 # Auf manchen Systemen:
 # set mail=/usr/spool/mail/`logname`

 # Konfigurieren des vi
setenv EXINIT = "set nu aw ws smd"

 # Setzen der Strg-Tasten fuer intr und kill
stty intr '^c' kill '^u'

 # Ausgabe des heutigen Datums
echo "**** Heute ist `date` ****"

 # Erinnerungs-Service
/usr/bin/calendar

 # Wechseln zum letzten work. dir. der letzten Sitzung
if (-f ~/.csh_zuletzt) then
 set workdir = `tail -1 ~/.csh_zuletzt`
endif
if ("$workdir" != "") then
 cd $workdir
endif
%
```

Wird der Inhalt von *.login* während einer UNIX-Sitzung geändert, ist damit nicht automatisch die entsprechende **csh**-Umgebung geändert. Dazu muß die **csh** zunächst den Inhalt von *.login* wieder neu lesen. Am einfachsten erreicht man dies unter Verwendung des builtin-Kommandos **source**:

**source ~/.login**

### 6.22.4 Die Datei *.logout*

Die Datei *.logout* kann für cleanup-Abarbeiten verwendet werden, die bei jedem Verlassen einer **csh** durchzuführen sind, wie z.B. Löschen aller Dateien in einem lokalen *tmp*-Directory. Eine andere Verwendung von *.logout* ist das Retten von Information über UNIX-Sitzungen hinweg. So könnte z.B. ein Benutzer mit

**history > ~/.csh\_history**

den History-Puffer in einer Datei hinterlegen und diese Kommandos beim erneuten Anmelden wieder verwenden.

Im nachfolgenden Beispiel für *.logout* wird der absolute Pfadname des momentanen Working-Directorys in der Datei *~/.csh\_zuletzt* gesichert, um bei einem erneuten Anmelden dieses wieder zum ersten Working-Directory zu machen, so daß der Benutzer in diesem Directory sofort weiter arbeiten kann. (Siehe auch vorheriges Kapitel bei *.login*):



```
% cat ~/.logout␣

Beispiel fuer das Aussehen der Datei $home/.logout

Speichern der Working-Directory
echo `pwd` > ~/.csh_zuletzt
%
```

## 6.23 Erweiterungen in manchen csh-Versionen

Manche **csh**-Versionen bieten noch zusätzliche Funktionalität an:

### Vordefinierte Shell-Variable **cdspell**

Wenn diese Variable **cdspell** gesetzt ist, versucht die **csh** Tippfehler beim Aufruf von **cd** mit einem nicht existierenden Directory zu erkennen und bessert diese selbst aus. Dazu fragt sie mit der Ausgabe eines existierenden und ähnlich lautenden Pfadnamens nach, ob eventuell zu diesem Directory gewechselt werden soll. Bei Eingabe von **n** findet dieser Directorywechsel nicht statt; jede andere Eingabe bewirkt den Wechsel zu diesem von der **csh** vorgeschlagenen Directory:

```
11 /user1/egon/cshueb>> cd /usr/bin␣
cd /usr/bin? ␣
ok
12 /usr/bin>> cd ~/chsueb␣
cd /user1/egon/cshueb? y␣
ok
13 /user1/egon/cshueb>>
```

### Scratch-Dateien

Ältere **csh**-Versionen unter BSD-UNIX erlaubten das Anlegen von sogenannten *Scratch*-Dateien. Dies sind temporäre Dateien, die vom System nach ein paar Tagen oder bei Speicherplatzmangel automatisch gelöscht werden. *scratch*-Dateien sind bei diesen **csh**-Versionen dadurch gekennzeichnet, daß ihr Name mit einem # beginnt. z.B. würde mit dem Aufruf

**banner Happy Birthday > #geburtstag**

eine *Scratch*-Datei *#geburtstag* angelegt.

### Das besondere Alias **shell**

Wenn ein Alias **shell** existiert, wird dessen Alias-Text beim Aufruf eines Shell-Skripts substituiert, so daß der Aufruf mit diesem Alias-Text stattfindet. Deswegen sollte das erste Wort des Alias-Textes der absolute Pfadname der entsprechenden Shell sein, z.B. **/bin/csh** oder **\$shell**:

```
29 /user1/egon/cshueb>> echo $shell ↵
/bin/csh
30 /user1/egon/cshueb>> alias shell $shell -x ↵ [-x bei Skript-Aufruf]
31 /user1/egon/cshueb>> cat ausgab ↵
#!/bin/csh
echo Das erste Argument ist $1
echo Das zweite Argument ist $2
echo Der Skriptname ist $0
32 /user1/egon/cshueb>> ausgab eins zwei drei ↵
echo Das erste Argument ist eins
Das erste Argument ist eins
echo Das zweite Argument ist zwei
Das zweite Argument ist zwei
echo Der Skriptname ist ausgab
Der Skriptname ist ausgab
32 /user1/egon/cshueb>>
```

## 6.24 Anwendungsbeispiel

Hier werden die Konstrukte der **csh** anhand von zwei umfangreicheren **csh**-Skripts wiederholt und im praktischen Einsatz gezeigt.

Die beiden nachfolgenden **csh**-Skripts *wahl* und *wahl\_info* sollen Umfragen zu beliebigen Themen automatisieren. Beim Aufruf des Skripts *wahl* wird zuerst eine Liste der momentan aktuellen Themen (z.B. Mittagessen, Betriebsausflug usw.) ausgegeben.

Nach der Auswahl des entsprechenden Themas werden dem Benutzer, falls die Wahl zu diesem speziellen Thema noch nicht abgeschlossen ist und sein Stimmenlimit noch nicht erschöpft ist, die Wahlmöglichkeiten angezeigt, z.B. für ein gemeinsames Mittagessen:

```
Schwarzer Loewe
Steakhaus
Knoedel-Willi
Wildschorsch
Rauchfang
El Sombrero
Goldner Drachen
```

Nach der entsprechenden Auswahl muß der Benutzer seine Wahl nochmals bestätigen, bevor diese dann festgehalten wird.

Der momentane Wahlstand kann mit dem **csh**-Skript *wahl\_info* erfragt werden.

Diese beiden **csh**-Skripts setzen im Directory, indem sie sich befinden, gewisse Dateien bzw. Subdirectories voraus. Wenn sich z.B. die beiden Skripts im Directory */user1/egon/umfrage* befinden, erwarten sie in diesem Directory eine Datei *.wahlthemen*, in der alle zur Verfügung stehenden Themen aufgelistet sind. Zu

jedem einzelnen Thema erwarten diese beiden Skripts ein Subdirectory, dessen Namen am Ende jeder Zeile in *.wahlthemen* nach einem Doppelpunkt angegeben ist; z.B. könnte *.wahlthemen* folgenden Inhalt haben:

```
Gemeinsames Mittagessen fuer Abteilung :mittagessen
Betriebsausflug :ausflug
Einarbeitung des halben Silvestertages :silvester
```

In diesen Subdirectories, wie z.B. *mittagessen* erwartet *wahl* eine Datei *.auswahl*, welche die Wahlmöglichkeiten zum betreffenden Thema beinhaltet, wie z.B.:

```
Schwarzer Loewe
Steakhaus
Knoedel Willi
Wildschorsch
Rauchfang
El Sombrero
Goldener Drachen
```

Über das Vorhandensein von bestimmten Dateien in den Subdirectories kann der Wahlmodus für das entsprechende Thema festgelegt werden:

- ▶ Existiert eine Datei *.geheim*, handelt es sich um eine geheime, ansonsten um eine nicht geheime Wahl.
- ▶ Existiert eine Datei *.geschlossen*, ist die Wahl bereits abgeschlossen; Existiert diese Datei nicht, können noch Stimmen abgegeben werden.
- ▶ Existiert eine Datei *.stimmzahl*, legt die darin enthaltene Zahl die erlaubten Wahlstimmen jedes Benutzers fest; existiert diese Datei nicht, so hat jeder Benutzer nur eine Stimme.

Auch legt das Skript *wahl* in diesen Subdirectories eigene Dateien an, um sich den Wahlstand zu merken:

- ▶ In *.gewaehlt* wird bei jeder Stimmabgabe der Name des entsprechenden Benutzers am Ende angehängt.
- ▶ In *.wahlurne* werden die abgegebenen Stimmen gesammelt.

Nachfolgend werden die notwendigen Aktionen des Wahlleiters gezeigt, wenn er die Wahl für den Themenkomplex »Mittagessen« ermöglichen möchte:

```
21 /user1/egon/umfrage>> cat .wahlthemen [↵] [muß erst erstellt werden]
Gemeinsames Mittagessen fuer Abteilung :mittagessen
Betriebsausflug :ausflug
Einarbeitung des halben Silvestertages :silvester
22 /user1/egon/umfrage>> mkdir mittagessen [↵]
23 /user1/egon/umfrage>> cd mittagessen [↵]
24 /user1/egon/umfrage/mittagessen>> cat .auswahl [↵] [muß erst erstellt
werden]

Schwarzer Loewe
Steakhaus
Knoedel Willi
Wildschorsch
Rauchfang
El Sombrero
Goldener Drachen
25 /user1/egon/umfrage/mittagessen>> cat >.stimmzahl [↵]
3 [↵] [Jeder Wähler hat 3 Stimmen]
Strg-D
26 /user1/egon/umfrage/mittagessen>> cd .. [↵]
27 /user1/egon/umfrage>>
```

Nun zu den beiden **csh**-Skripten *wahl* und *wahl\_info*:

```
28 /user1/egon/umfrage>> cat wahl [↵]
#!/bin/csh -f
#
wahl - Skript, welches Abstimmungen jeglicher Art
automatisiert.
#

set dir = $0
set dir = $dir:h
if ("$dir" == "$0") set dir = "."

set themen_datei = $dir/.wahlthemen

if (-e $themen_datei) then
 @ themen_zahl = `cat $themen_datei | wc -l`
else
 @ themen_zahl = 0
endif

switch ($themen_zahl)
case 0:
 echo "Es steht zur Zeit kein Thema zur Wahl"
 exit 1
case 1: echo "Es steht zur Zeit nur ein Thema zur Wahl"
 echo " "; echo " "
```

```
echo "-----"
nl -ba $themen_datei
echo "-----"
breaksw
default:
echo " "; echo " "
echo "Zur Wahl stehen folgende Themen:"
echo "-----"
nl -ba $themen_datei
echo "-----"
set thema_wahl = ""
while ("$thema_wahl" == "")
 echo -n "Bitte waehlen Sie die entsprechende Nummer: "
 set thema_wahl = $<
 @ nr = $themen_zahl + 1 - $thema_wahl
 if ($nr > 0) then
 set nonomatch
 set thema_zeile = `tail - $nr $themen_datei | line`
 set thema_wahl = `echo $thema_zeile | cut -d":" -f2`
 set thema_wahl = ($dir/$thema_wahl)
 set thema_zeile = `echo $thema_zeile | cut -d":" -f1`
 else
 echo " "; echo " "
 echo "Erlaubte Themen sind:"
 echo "-----"
 nl -ba $themen_datei
 echo "-----"
 set thema_wahl = ""
 endif
end
end
set thema = $thema_wahl:t
endsw

if (! -e $dir/$thema) then
 echo "Wahldirectory --- $thema --- nicht vorhanden"
 exit (1)
endif

cd $dir/$thema

if (-e .geschlossen) then
 echo
 echo "Wahl fuer"
 echo " --- $thema_zeile ---"
 echo "bereits abgeschlossen."
 echo "Rufen Sie --- wahl_info --- auf, um Wahlergebnisse zu erfahren"
 exit 0
endif

if (-e .geheim) then
 echo
```

```
 echo "Wahl fuer"
 echo " --- $thema_zeile ---"
 echo "ist eine geheime Wahl."
 set ist_geheim = 1
else
 set ist_geheim = 0
endif

if (-e .stimmzahl) then
 set stimm_zahl = (`cat .stimmzahl`)
else
 set stimm_zahl = 1
endif

set waehler = `logname`
touch .gewaehlt
set wieoft_gewaehlt = \
 (`fgrep $waehler .gewaehlt | wc -l `)
if ($wieoft_gewaehlt >= $stimm_zahl) then
 echo "Tut mir leid, aber sie haben schon"
 echo "alle Ihre Stimmen ($stimm_zahl) zu"
 echo " --- $thema_zeile ---"
 echo "bereits abgegeben"
 exit 0
endif

set auswahl = .auswahl
if (-e $auswahl) then
 @ auswahl_zahl = `cat $auswahl | wc -l`
else
 @ auswahl_zahl = 0
endif

switch ($auswahl_zahl)
case 0:
 echo "Es ist zur Zeit keinerlei Wahl moeglich"
 exit 1
case 1: echo "Es steht zur Zeit nur eine Auswahl zur Verfuegung"
 nl -ba $auswahl
 breaksw
default:
 echo " "; echo " "
 echo "Folgende Wahlmoeglichkeiten besitzen Sie:"
 echo "-----"
 nl -ba $auswahl
 echo "-----"
 set wahl = ""
 while (" $wahl" == "")
 echo -n "Bitte waehlen Sie die entsprechende Nummer: "
 set wahl = $<
 @ nr = $auswahl_zahl + 1 - $wahl
```

```
 if ($nr > 0) then
 set nonomatch
 set wahl_zeile = `tail - $nr $auswahl | line`
 set wahl = `echo $wahl_zeile | cut -d":" -f2`
 set wahl_zeile = `echo $wahl_zeile | cut -d":" -f1`
 echo "Sie haben"
 echo " --- $wahl_zeile ---"
 echo " gewählt."
 echo -n "Ist das richtig (j /n) ? "
 if ($< !~ [jJ]*) set wahl = ""
 else
 echo " "; echo " "
 echo "Folgende Wahlmöglichkeiten besitzen Sie:"
 echo "-----"
 nl -ba $auswahl
 echo "-----"
 set wahl = ""
 endif
 end
end

echo $waehler >>!.gewaehlt

if ($ist_geheim == 1) then
 echo "`date` | $wahl" >>!.wahlurne
 echo "Vielen Dank. Ihre Stimme wurde geheim festgehalten"
else
 echo "`date` | $wahl | $waehler" >>!.wahlurne
 echo "Vielen Dank. Ihre Stimme wurde festgehalten"
endif

29 /user1/egon/umfrage>> chmod +x wahl
30 /user1/egon/umfrage>> cat wahl_info
#!/bin/csh -f
#
wahl_info - Skript, welches Informationen zum Stand von
laufenden bzw. abgeschlossenen Abstimmungen
liefert.
#

set dir = $0
set dir = $dir:h
if (" $dir" == "$0") set dir = "."

set themen_datei = $dir/.wahlthemen

if (-e $themen_datei) then
 @ themen_zahl = `cat $themen_datei | wc -l`
else
 @ themen_zahl = 0
endif
```

```
switch ($themen_zahl)
case 0:
 echo "Es steht zur Zeit kein Thema zur Wahl"
 exit 1
case 1: echo "Es steht zur Zeit nur ein Thema zur Wahl"
 echo " "; echo " "
 echo "-----"
 nl -ba $themen_datei
 echo "-----"
 breaksw
default:
 echo " "; echo " "
 echo "Zur Wahl stehen folgende Themen:"
 echo "-----"
 nl -ba $themen_datei
 echo "-----"
 set thema_wahl = ""
 while ("$thema_wahl" == "")
 echo -n "Bitte waehlen Sie die entsprechende Nummer: "
 set thema_wahl = $<
 @ nr = $themen_zahl + 1 - $thema_wahl
 if ($nr > 0) then
 set nonomatch
 set thema_zeile = `tail -n $nr $themen_datei | line`
 set thema_wahl = `echo $thema_zeile | cut -d":" -f2`
 set thema_wahl = ($dir/$thema_wahl)
 set thema_zeile = `echo $thema_zeile | cut -d":" -f1`
 else
 echo " "; echo " "
 echo "Erlaubte Themen sind:"
 echo "-----"
 nl -ba $themen_datei
 echo "-----"
 set thema_wahl = ""
 endif
 end
 set thema = $thema_wahl:t
endsw

if (! -e $dir/$thema) then
 echo "Wahl directory --- $thema --- nicht vorhanden"
 exit (1)
endif

cd $dir/$thema

echo " "
echo "Wahl fuer"
echo " --- $thema_zeile ---"
if (-e .geschlossen) then
 echo "bereits abgeschlossen."
```



```
else
 echo "noch im Gange."
endif

if (-e .wahlurne) then
 echo " "
 echo "Wahlstand am `date`"
 echo "-----"
 cut -d"| " -f2 .wahlurne | sort | uniq -c | sort -rn
 echo "-----"; echo " "
else
 echo "Keine Wahlurne vorhanden"
 exit (1)
endif

if (! -e .geheim) then
 echo "Die einzelnen Waehler gaben folgende Stimme ab:"
 echo "-----"
 cut -d"| " -f2- .wahlurne | sort | uniq -c
 echo "-----"
endif
31 /user1/egon/umfrage>> chmod +x wahl_info
32 /user1/egon/umfrage>>
```

Nachfolgend werden nun Beispiele für den Ablauf der beiden **csH**-Skripts *wahl* und *wahl\_info* gegeben:

```
32 /user1/egon/umfrage>> wahl
```

Zur Wahl stehen folgende Themen:

```

1 Gemeinsames Mittagessen fuer Abteilung :mittagessen
2 Betriebsausflug :ausflug
3 Einarbeitung des halben Silvestertages :silvester

```

Bitte waehlen Sie die entsprechende Nummer: 1

Folgende Wahlmoeglichkeiten besitzen Sie:

```

1 Schwarzer Loewe
2 Steakhaus
3 Knoedel Willi
4 Wildschorsch
5 Rauchfang
6 El Sombrero
7 Goldener Drachen

```

Bitte waehlen Sie die entsprechende Nummer: 3

```
Sie haben
--- Knoedel Willi ---
gewaehlt.
Ist das richtig (j /n) ? j
Vielen Dank. Ihre Stimme wurde festgehalten
33 /user1/egon/umfrage>> wahl_info
```

Zur Wahl stehen folgende Themen:

```

1 Gemeinsames Mittagessen fuer Abteilung :mittagessen
2 Betriebsausflug :ausflug
3 Einarbeitung des halben Silvestertages :silvester

```

Bitte waehlen Sie die entsprechende Nummer: 1

```
Wahl fuer
--- Gemeinsames Mittagessen fuer Abteilung ---
noch im Gange.
```

Wahlstand am Thu Jul 25 09:14:12 EDT 1997

```

3 Wildschorsch
2 Goldener Drachen
1 Knoedel Willi

```

Die einzelnen Waehler gaben folgende Stimme ab:

```

2 Goldener Drachen | emil
1 Knoedel Willi | egon
2 Wildschorsch | egon
1 Wildschorsch | emil

```

```
34 /user1/egon/umfrage>>
```



# 7 **bash – Bourne Again Shell**

*Heute kennt man von allem den Preis,  
von nichts den Wert.*

*Wilde*

Die **bash** ist ein Nachfolger der Bourne-Shell. Sie ist als freie Software verfügbar. Unter Linux, das immer weitere Verbreitung findet, ist sie standardmäßig als Benutzer-Shell eingerichtet. Soll unter Linux eine andere Shell als Default-Shell eingerichtet werden, so ist dies mit dem Kommando **chsh** (*change shell*) möglich.

Die **bash** enthält Elemente aus den anderen Shells **sh**, **ksh**, **csh** und **tcsh** (Erweiterte Form der **csh**). Sie bietet jedoch auch neue Konstrukte an, die in keiner der anderen Shells vorhanden sind. Nachfolgend wird die **bash** kurz vorgestellt, wobei vor allen Dingen die Unterschiede und Neuheiten der **bash** gegenüber der Bourne-Shell (siehe auch Kapitel 4) besonders hervorgehoben werden. Um weitere Informationen zur **bash** zu erhalten, empfiehlt es sich, entweder **man bash** oder **info -f /usr/info/bash** unter Linux aufzurufen.

## 7.1 **Wesentliche Unterschiede der bash zur Bourne-Shell**

Die **bash** verwendet im großen und ganzen die gleiche Grammatik, Parameter-Substitution, Dateinamenexpansion, Ein-/Ausgabumlenkung und Quoting wie die Bourne-Shell. Nachfolgend wird ein Überblick über die wichtigsten Unterschiede der **bash** zur Bourne-Shell gegeben.

### **History-Mechanismus**

Zugriffe auf vorherige Kommandozeilen erfolgen in der **bash** durch einfaches Drücken der *Pfeil-oben*-Taste. Hat man sich in der History-Liste zu weit hoch bewegt, kann man sich mit der *Pfeil-unten*-Taste wieder weiter nach unten in der History-Liste bewegen.

### Editieren von Kommandozeilen

Zum Editieren von Kommandozeilen stehen in der **bash** eine Vielzahl von **emacs**- bzw. **vi**-Kommandos zur Verfügung. Die wichtigsten Editier-Tasten und -Kommandos sind:

- ▶ *Pfeil-links, Pfeil-rechts*      Cursor nach links / rechts bewegen
- ▶ *POS1, ENDE*                  Cursor auf Anfang / Ende der Zeile bewegen
- ▶ *Strg-A, Strg-E*                Cursor auf Anfang / Ende der Zeile bewegen
- ▶ *Alt-B, Alt-F*                  Cursor um ein Wort nach links / rechts bewegen
- ▶ *Backspace, ENTF*            Zeichen links / rechts löschen
- ▶ *ALT-D*                          Wort löschen
- ▶ *Strg-K*                        Bis Zeilenende löschen
- ▶ *Strg-L*                        Bildschirm löschen

### Vervollständigung von Namen mit TAB-Taste

Die **bash** versucht beim Drücken der Tabulator-Taste eine Vervollständigung für aktuelles Wort (Text vor Cursor). Beginnt Wort mit **\$**, versucht sie einen existierenden Variablennamen hieraus zu machen. Beginnt Wort mit **~**, versucht sie einen existierenden Benutzernamen hieraus zu machen. Beginnt Wort mit **@**, versucht sie einen existierenden Hostnamen hieraus zu machen. Trifft keine dieser vorherigen Bedingungen zu, versucht sie hieraus einen existierenden Alias- oder Funktionsnamen zu machen. Sollte auch dies fehlschlagen, versucht sie einen existierenden Pfadnamen aus dem Wort zu machen. Schlagen alle Expanderungsversuche fehl, so erklingt die Terminalglocke. Ein erneutes Drücken der TAB-Taste zeigt dann alle möglichen Vervollständigungsmöglichkeiten an; erneutes Klingeln weist darauf hin, wenn dies nicht möglich ist.

### Ausrufezeichen ! bei Pipes

Das Ausrufezeichen negiert den Rückgabewert einer Pipe. Das Schlüsselwort **!** ist sehr nützlich, wenn ein **if**-Kommando nur dann auszuführen ist, wenn das angegebene Kommando (oder Pipeline) nicht erfolgreich war.

### csh-Dateinamenexpansion {pattern1,pattern2,...}

**bash** expandiert z.B. `{add,mult}.c` zu `add.c mult.c`.

### ksh-Kommandosubstitution \$(kdo liste)

Neben der alten Kommandosubstitution der Bourne-Shell ``kdo liste`` bietet die **bash** auch die neue Kommandosubstitution `$(kdo liste)` der Korn-Shell an.

**Länge eines Strings `${#var}`**

Die Angabe `${#var}` liefert die Länge des Strings in der Variablen `var`.

**ksh-Parametersubstitution `${var##pattern}` und `${var%%pattern}`**

`${var##pattern}` liefert Wert von `var` ohne größtmöglichen durch `pattern` abgedeckten linken Teilstring und `${var%%pattern}` liefert Wert von `var` ohne größtmöglichen durch `pattern` abgedeckten rechten Teilstring. Beide Konstruktionen sind in Bourne-Shell nicht vorhanden.

**Automatische Vererbung von bestimmten Variablen**

Variablen, die in der ursprünglichen Environment der **bash** vorhanden sind, werden automatisch an Kindprozesse (Subshells) exportiert. In der Bourne-Shell müssen solche Variable explizit mit **export** zum Export markiert werden.

**Eingeschränkte Verwendung der Variablen IFS**

Die Variable **IFS** wird nur verwendet, um Resultate von Expandierungen zu trennen, und nicht für die Trennung aller Wörter. Dies war ein Sicherheitsloch in der Bourne-Shell.

**Gleiche Variablen- und Funktionsnamen sind erlaubt**

Dies war in der Bourne-Shell nicht möglich.

**Deklaration von funktionslokalen Variablen ist möglich**

Dies löst viele Probleme, die beim Schreiben von rekursiven Funktionen in der Bourne-Shell auftraten.

**Option `noclobber`**

Wenn **noclobber** gesetzt ist, können mit Ausgabeumlenkung existierende Dateien nicht überschrieben werden.

**Builtin-Kommandos können durch Funktionen ersetzt werden**

In der Funktionsdefinition kann auf das entsprechende builtin-Kommando mit den beiden neuen builtin-Kommandos **builtin** und **command** zugegriffen werden.

**Neues builtin-Kommando `command`**

**command** ermöglicht das wahlweise Ausschalten von Funktionen beim Aufruf von Kommandos.

### Ein- und Ausschalten eigener builtin-Kommandos mit **enable**

### Exportieren von Funktionen an Kindprozesse (Subshells)

Funktionen können über die Umgebung (Environment) an Kindprozesse (Subshells) exportiert werden.

### Kleinere Unterschiede beim **read**-Kommando

Mit dem **read** der **bash** ist es möglich, Zeilen zu lesen, die mit Backslash \ enden, wenn die Option **-r** dabei angegeben ist. Auch schreibt das **read** der **bash** eine gelesene Zeile in die Variable **REPLY**, wenn keine Variablen beim **read**-Aufruf angegeben sind.

### **return** beendet Skripts, die mit **.** (Punkt) oder **source** gelesen werden

### **umask** kennt symbolischen Modus wie **chmod**

### Kleinere Unterschiede beim **test**-Kommando

## 7.2 Optionen und Argumente für die **bash**

**bash** [*optionen*] [*datei*]

Neben den typischen Einzeichen-Optionen, die auch mit dem Kommando **set** gesetzt werden können, sind zusätzlich auch noch die Optionen der Bourne-Shell **-c**, **-i** und **-s** verfügbar.

Daneben bietet die **bash** noch weitere Optionen an, die vor den entsprechenden Einzeichen-Optionen anzugeben sind:

| Option               | Beschreibung                                                                                                                                                                                                                        |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-norc</b>         | Lesen der Initialisierungs-Datei <b>~/.bashrc</b> beim Start einer interaktiven <b>bash</b> unterdrücken. Voreinstellung ist, daß diese Option bei einem <b>sh</b> -Aufruf automatisch gesetzt wird.                                |
| <b>-noprofile</b>    | Lesen aller Startup-Dateien ( <b>/etc/profile</b> , <b>~/.bash_profile</b> , <b>~/.bash_login</b> oder <b>~/.profile</b> ) unterdrücken, wenn <b>bash</b> als Login-Shell gestartet wird.                                           |
| <b>-rcfile datei</b> | Lesen der Initialisierungs-Datei <i>datei</i> (anstelle von <b>~/.bashrc</b> ) beim Start einer interaktiven <b>bash</b> .                                                                                                          |
| <b>-version</b>      | Ausgeben der Versionsnummer beim Starten der <b>bash</b> .                                                                                                                                                                          |
| <b>-quiet</b>        | keine Information (wie Version) beim Start ausgeben.                                                                                                                                                                                |
| <b>-login</b>        | Shell soll sich so verhalten, als ob sie direkt vom <b>login</b> -Kommando aufgerufen wurde. Falls die aktuelle Login-Shell durch eine <b>bash</b> -Login-Shell ersetzt werden soll, muß <b>exec bash -login</b> aufgerufen werden. |

| Option                   | Beschreibung                                                                             |
|--------------------------|------------------------------------------------------------------------------------------|
| <b>-nobraceexpansion</b> | Dateinamenexpandierung der Form {wort <sub>1</sub> ,wort <sub>2</sub> ,...} ausschalten. |
| <b>-nolineediting</b>    | Kommandozeilen-Editierung ausschalten.                                                   |
| <b>-posix</b>            | <b>bash</b> soll sich wie eine POSIX 1003.2-Shell verhalten.                             |

Sind bei einem **bash**-Aufruf nach der Auswertung der Optionen noch weitere Argumente vorhanden, und die Optionen **-c** und **-s** waren nicht angegeben, wird angenommen, daß es sich beim ersten Argument um den Dateinamen eines zu startenden Shellskripts handelt. Die restlichen Argumente sind dann die Positionsparameter **\$1**, **\$2** usw. Die **bash** startet dann das entsprechende Shellskript und führt die darin enthaltenen Kommandos aus. Der exit-Status eines solchen **bash**-Aufrufs ist der exit-Status des letzten Kommandos im Shellskript.

## 7.3 Einfache Kommandos, Pipelines und Listen

Einfache Kommandos, Pipelines und Listen sind in der **bash** weitgehend identisch zur Bourne-Shell definiert.

Ein Unterschied zur Bourne-Shell ist das Schlüsselwort **!** (Ausrufezeichen), mit dem der exit-Status einer Pipe negiert werden kann. Das Schlüsselwort **!** ist sehr nützlich, wenn ein **if**-Kommando nur dann auszuführen ist, wenn das angegebene Kommando (oder Pipeline) nicht erfolgreich war.

## 7.4 Kommandos zur Ablaufsteuerung

(*kdoliste*)      *kdoliste* wird in einer Subshell ausgeführt; identisch zur Bourne-Shell.

{*kdoliste*; }      *kdoliste* wird in aktuellen Shell (nicht Subshell) ausgeführt; identisch zur Bourne-Shell.

**if-Anweisung** (identisch zur Bourne-Shell)

```

if if_kdoliste1
then
 then_kdoliste1
[elif if_kdoliste2
then
 then_kdoliste2
 :]
[else
 else_kdoliste]
fi
```



**case-Anweisung** (identisch zur Bourne-Shell)

```
case wort in
 pattern1) kdoliste1;;
 pattern2) kdoliste2;;
 :
 patternn) kdolisten;;
esac
```

**while-Schleife** (identisch zur Bourne-Shell)

```
while kdoliste1
do
 kdoliste2
done
```

**until-Schleife** (identisch zur Bourne-Shell)

```
until kdoliste1
do
 kdoliste2
done
```

**for-Schleife** (identisch zur Bourne-Shell)

```
for laufvariable [in wort1 ... wortn]
do
 kdoliste
done
```

**select-Anweisung** (identisch zur Korn-Shell)

```
select variable [in wort1 ... wortn]
do
 kdoliste
done
```

**Funktionen** ähnlich zu den Funktionen in der Korn-Shell

```
[function] name() { kdoliste; }
```

definiert eine Funktion mit Namen *name*. In der Funktion können mit dem builtin-Kommando **local** lokale Variablen definiert werden.

## 7.5 Kommentare

Beginnt ein Wort mit **#**, werden dieses Wort und alle nachfolgenden Zeichen dieser Zeile von der Shell als Kommentar interpretiert und als solcher von ihr ignoriert.

Die Sonderbedeutung des Kommentarzeichens **#** ist in einer interaktiven Shell ausgeschaltet, wenn die Option **interactive-comments** ausgeschaltet ist (**set +o interactive-comments**)

## 7.6 Quoting

Das Quoting der **bash** entspricht weitgehend dem Quoting der Bourne-Shell.

Neben der Kommandosubstitution der Bourne-Shell ``kommando`` wird jedoch auch die neu hinzugekommene Kommandosubstitution `$(kommando)` innerhalb von `"..."` ausgewertet.

## 7.7 Parameter und Variablen

Wie die Bourne-Shell kennt die **bash** Positionsparameter und Shellvariablen. Bezüglich der Wahl von benutzerdefinierten Variablennamen gelten die gleichen Regeln wie in der Bourne-Shell. Wie in der Bourne-Shell kann einer Variablen mit

```
variable=[wert]
```

ein Wert zugewiesen werden. Ist kein *wert* angegeben, wird *variable* der Null-String zugewiesen. Für den angegebenen *wert* wird Parameter-, Kommandosubstitution und Dateinamenexpansion durchgeführt.

Anders als in der Bourne-Shell wird für den angegebenen Wert auch Tilde-Expansion und eventuell eine arithmetische Auswertung durchgeführt. Ist für die *variable* das Attribut **-i** gesetzt, findet selbst dann eine arithmetische Auswertung statt, wenn **\${..}** nicht verwendet wurde.

### 7.7.1 Positionsparameter

Wie in Korn-Shell kann auf Positionsparameter mit **\$1**, **\$2**, ..., **\$9**, **\${10}**, **\${11}**, ... zugegriffen werden.

Mit dem builtin-Kommando **set** können wie in der Bourne-Shell die Positionsparameter neu gesetzt werden. Wird eine Shell-Funktion aufgerufen, ersetzen die beim Aufruf angegebenen Argumente für die Dauer des Funktionsaufrufs die momentanen Positionsparameter.

## 7.7.2 Automatische Variablen

Die folgenden vordefinierten Variablen werden ständig neu von der **bash** gesetzt, wobei auf die Werte dieser automatischen Variablen wie in der Bourne-Shell wieder durch Voranstellen von **\$** zugegriffen werden kann.

| Variable  | Beschreibung                                                                                                                                                                                           |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>#</b>  | Anzahl der gesetzten Positionsparameter                                                                                                                                                                |
| <b>-</b>  | Momentan gesetzte Shell-Optionen                                                                                                                                                                       |
| <b>?</b>  | Exit-Status des letzten Vordergrund-Kommandos                                                                                                                                                          |
| <b>\$</b> | PID der aktuellen Shell; in (...) ist es die PID der aktuellen Shell, und nicht der Subshell.                                                                                                          |
| <b>!</b>  | PID des letzten Hintergrund-Kommandos                                                                                                                                                                  |
| <b>*</b>  | Alle Positionsparameter als <u>ein</u> String:<br>"\$*" entspricht "\$1 \$2 \$3 ..."                                                                                                                   |
| <b>@</b>  | Alle Positionsparameter als <u>einzelne</u> Strings:<br>"\$@" entspricht "\$1" "\$2" "\$3" ...                                                                                                         |
| <b>0</b>  | Name der Shell oder des Shellskripts. Wurde <b>bash</b> mit Option <b>-c</b> gestartet, befindet sich in <b>\$0</b> das erste Argument, nachdem der nach <b>-c</b> angegebene String ausgeführt wurde. |
| <b>_</b>  | (Unterstrich) Letztes Argument (nach Expandierung) des vorherigen Kommandos.                                                                                                                           |

## 7.7.3 Shellvariablen

Die folgenden vordefinierten Variablen werden ständig neu von der **bash** gesetzt, wobei auf die Werte dieser Variablen wieder durch Voranstellen von **\$** zugegriffen werden kann.

| Variable            | Beschreibung                                                                                                                                  |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>BASH</b>         | voller Pfadname der <b>bash</b>                                                                                                               |
| <b>BASH_VERSION</b> | Versionsnummer der <b>bash</b>                                                                                                                |
| <b>EUID</b>         | Effektive User-ID des <b>bash</b> -Benutzers                                                                                                  |
| <b>HISTCMD</b>      | History-Nummer des aktuellen Kommandos. Wird <b>HISTCMD</b> gelöscht, verliert es diese Bedeutung und kann nicht erneut eingeschaltet werden. |
| <b>HOSTTYPE</b>     | Typ der Maschine                                                                                                                              |

| Variable              | Beschreibung                                                                                                                                                                                                                                                         |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>LINENO</b>         | (wie in ksh) enthält die aktuelle Zeilennummer in einem Skript oder in einer Funktion. Wird <b>LINENO</b> gelöscht, verliert es diese Bedeutung und kann nicht erneut eingeschaltet werden.                                                                          |
| <b>OLDPWD</b>         | (wie in ksh) Vorheriges Working-Directory                                                                                                                                                                                                                            |
| <b>OPTIND</b>         | Argument-Nummer der nächsten auszuwertenden Option beim <b>getopts</b> -Kommando                                                                                                                                                                                     |
| <b>OPTARG</b>         | das zu einer Option angegebene Argument, wenn <b>getopts</b> eine solche Option liest                                                                                                                                                                                |
| <b>OSTYPE</b>         | Name des Betriebssystems                                                                                                                                                                                                                                             |
| <b>PPID</b>           | (wie in ksh) PID der Elternshell ( <b>P</b> arent <b>P</b> rocess <b>I</b> D)                                                                                                                                                                                        |
| <b>PROMPT_COMMAND</b> | Jedesmal, bevor <b>bash</b> Primär-Prompt ausgibt, führt sie eventuell in dieser Variable angegebene Kommandos aus.                                                                                                                                                  |
| <b>PWD</b>            | (wie in ksh) Aktuelles Working-Directory                                                                                                                                                                                                                             |
| <b>RANDOM</b>         | (wie in ksh) liefert bei jedem Zugriff eine neue Zufallszahl zwischen 0 und 32767. Zuweisen eines Werts an diese Variable setzt den Zufallszahlengenerator neu. Wird <b>RANDOM</b> gelöscht, verliert es diese Bedeutung und kann nicht erneut eingeschaltet werden. |
| <b>REPLY</b>          | (wie in ksh) enthält die eingegebenen Zeichen, wenn bei <b>select</b> oder bei <b>read</b> keine Argumente angegeben sind.                                                                                                                                           |
| <b>SECONDS</b>        | enthält die seit <b>bash</b> -Aufruf verstrichenen Sekunden. Wird <b>SECONDS</b> gelöscht, verliert es diese Bedeutung und kann nicht erneut eingeschaltet werden.                                                                                                   |
| <b>SHLVL</b>          | wird bei jedem Start einer <b>bash</b> um 1 erhöht und beim Verlassen wieder um 1 dekrementiert. So läßt sich Tiefe (Anzahl) der Subshells erfragen, die man gestartet hat.                                                                                          |
| <b>UID</b>            | Reale User-ID des <b>bash</b> -Benutzers.                                                                                                                                                                                                                            |

Die folgenden vordefinierten Variablen werden von der **bash** nur gelesen und ausgewertet. Manchen weist die **bash** voreingestellte Werte zu.

| <i>Variable</i>                        | <i>Beschreibung</i>                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>allow_null_glob_expansion</b>       | Wenn gesetzt, wird bei einer Dateinamenexpansion, die keinerlei Dateien abdeckt, nicht der angegebene String, sondern der Null-String verwendet.                                                                                                                                                                                                                                                                                                          |
| <b>cdable_vars</b>                     | Wenn gesetzt, wird bei Angabe eines Namens bei <b>cd</b> , der kein Directory ist, angenommen, daß es sich dabei um einen Namen einer Variable handelt, die das entsprechende Directory enthält.                                                                                                                                                                                                                                                          |
| <b>CDPATH</b>                          | Suchdirectories für <b>cd</b> .                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>command_oriented_history</b>        | Wenn gesetzt, versucht <b>bash</b> alle mehrzeiligen Kommandos als eine Zeile in der History aufzunehmen, um das Editieren solcher Kommandos zu erleichtern.                                                                                                                                                                                                                                                                                              |
| <b>ENV</b>                             | (wie in ksh) Pfadname der Environment-Datei, die <b>bash</b> lesen soll, wenn sie Skript ausführt.                                                                                                                                                                                                                                                                                                                                                        |
| <b>FCEDIT</b>                          | (wie in ksh) Pfadname des builtin-Editors, der beim Kommando <b>fc</b> zu verwenden ist.                                                                                                                                                                                                                                                                                                                                                                  |
| <b>FIGNORE</b>                         | Mit Doppelpunkt getrennte Liste von Dateinamen-Endungen, die bei der Dateinamen-Vervollständigung zu ignorieren sind. Dateinamen mit diesen Endungen werden dabei nicht angezeigt.                                                                                                                                                                                                                                                                        |
| <b>glob_dot_filenames</b>              | Wenn gesetzt, deckt <b>bash</b> bei der Dateinamenexpansion auch Dateinamen ab, die mit . (Punkt) beginnen.                                                                                                                                                                                                                                                                                                                                               |
| <b>histchars</b>                       | Definiert die History-Metazeichen. Das erste Zeichen ersetzt ! und das zweite Zeichen ersetzt ^. Das optionale dritte Zeichen ersetzt das voreingestellte History-Kommentarzeichen #. Zeichen nach dem History-Kommentarzeichen werden nicht in History aufgenommen.                                                                                                                                                                                      |
| <b>HISTCONTROL<br/>history_control</b> | Wenn auf <b>ignorespace</b> gesetzt, werden Zeilen, die mit Leer- oder Tabzeichen beginnen, nicht in History übernommen.<br><br>Wenn auf <b>ignoredups</b> gesetzt, werden identische Zeilen, die nacheinander eingegeben werden, nicht in History übernommen.<br><br>Wenn auf <b>ignoreboth</b> gesetzt, werden weder Zeilen, die mit Leer- oder Tabzeichen beginnen, noch identische Zeilen, die nacheinander eingegeben werden, in History übernommen. |
| <b>HISTFILE</b>                        | (wie in ksh) Pfadname der History-Datei.<br>(Voreinstellung: <b>~/bash_history</b> ).                                                                                                                                                                                                                                                                                                                                                                     |

| Variable                        | Beschreibung                                                                                                                                                                                                                                                                              |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>HISTFILESIZE</b>             | legt fest, wie viele Zeilen maximal in History-Datei aufzuheben sind (Voreinstellung: 500).                                                                                                                                                                                               |
| <b>HISTSIZE</b>                 | (wie in ksh) legt fest, wie viele Kommandos in History-Datei aufzuheben sind (Voreinstellung: 500).                                                                                                                                                                                       |
| <b>HOME</b>                     | Home-Directory des Benutzers.                                                                                                                                                                                                                                                             |
| <b>hostname_completion_file</b> | enthält Namen einer Datei, die neben <i>/etc/hosts</i> bei Hostnamen-Vervollständigung zu verwenden ist.                                                                                                                                                                                  |
| <b>HOSTFILE</b>                 |                                                                                                                                                                                                                                                                                           |
| <b>IFS</b>                      | Wort-Trennzeichen (Voreinstellung: Leer-, Tab- und Neuezeile-Zeichen).                                                                                                                                                                                                                    |
| <b>IGNOREEOF</b>                | wenn definiert, legt ihr Wert die Anzahl der EOFs ( <i>Strg-D</i> ) fest, die unmittelbar nacheinander einzugeben sind, bevor interaktive <b>bash</b> sich beendet (Voreinstellung: 10).                                                                                                  |
| <b>INPUTRC</b>                  | Wenn gesetzt, so wird anstelle von <i>~/inputrc</i> die dort enthaltene Datei verwendet.                                                                                                                                                                                                  |
| <b>MAIL</b>                     | Pfadname der mailbox-Datei.                                                                                                                                                                                                                                                               |
| <b>MAILCHECK</b>                | Zeitintervall (in Sekunden), in dem das Ankommen neuer mail zu prüfen ist (Voreinstellung: 60).                                                                                                                                                                                           |
| <b>MAILPATH</b>                 | Pfadnamen von mailbox-Dateien. Die einzelnen mailbox-Pfadnamen sind bei der Angabe mit <i>:</i> zu trennen. Die zu einem Pfadnamen gehörige Meldung kann, mit <i>?</i> abgetrennt, nach diesem angegeben werden. Ein <i>\$_</i> im Text wird durch den aktuellen mail-Dateinamen ersetzt. |
| <b>MAILWARNING</b>              | Wenn gesetzt und wenn auf eine mailbox-Datei seit der letzten Prüfung zugegriffen wurde, wird "The mail in <i>mailbox-datei</i> has been read" ausgegeben.                                                                                                                                |
| <b>no_exit_on_failed_exec</b>   | Ist diese Variable gesetzt, beendet sich eine nicht-interaktive Shell nicht, wenn sie die bei <b>exec</b> angegebene Datei nicht ausführen kann. Eine interaktive Shell beendet sich nie beim einem mißglückten <b>exec</b> -Aufruf.                                                      |
| <b>noclobber</b>                | (wie in ksh) Ausgabeumlenkung in existierende Dateien unterbinden.                                                                                                                                                                                                                        |
| <b>nolinks</b>                  | Wenn gesetzt, folgt <b>bash</b> beim Wechseln in andere Directories (siehe auch Option <b>-P</b> beim <b>set</b> -Kommando) keinen symbolischen Links.                                                                                                                                    |
| <b>notify</b>                   | Wenn gesetzt, wird Status von beendeten Hintergrund-Jobs sofort und nicht erst bei Ausgabe des nächsten Prompts gemeldet.                                                                                                                                                                 |

| Variable      | Beschreibung                                                                                                                                                                                                    |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>OPTERR</b> | Wenn auf 1 gesetzt, gibt <b>bash</b> Fehlermeldungen aus, die aus <b>getopts</b> -Aufruf resultieren. (Voreinstellung: wird beim Start einer <b>bash</b> oder eines Skripts immer auf 1 gesetzt).               |
| <b>PATH</b>   | Suchdirectories für Programme. Die einzelnen Directories sind dabei in der gewünschten Such-Reihenfolge anzugeben und mit : voneinander zu trennen. Ein leeres Directory steht dabei für das Working-Directory. |
| <b>PS1</b>    | Primär-Promptstring (Voreinstellung: " <b>bash</b> \$ ").                                                                                                                                                       |
| <b>PS2</b>    | Sekundär-Promptstring (Voreinstellung: "> ").                                                                                                                                                                   |
| <b>PS3</b>    | (wie in ksh) Promptstring für die Auswahl beim Kommando <b>select</b> .                                                                                                                                         |
| <b>PS4</b>    | (wie in ksh) Debugging-Promptstring für Shell-Skripts (Voreinstellung: "+ ").                                                                                                                                   |
| <b>TMOUT</b>  | (wie in ksh) legt eine Zeitperiode (in Sekunden) fest, in der immer ein Kommando einzugeben ist. Erfolgt in dieser Zeitperiode keine Eingabe, beendet sich die <b>bash</b> automatisch.                         |

## 7.8 Substitutionen und Expandierungen

Expandierungen werden in der Kommandozeile durchgeführt, nachdem sie in einzelne Wörter aufgeteilt wurde.

### 7.8.1 Dateinamenexpandierungen mit {...}

Bei Angabe von `{wort1,wort2,...}` wird jedes der in `{..}` mit Komma getrennten Worte eingesetzt; z.B. würde `prog.{c,o,txt}` die Dateien `prog.c`, `prog.o` und `prog.txt` und `{add1,subst}.c` die Dateien `add1.c` und `subst.c` abdecken, wenn diese existieren. Diese Form der Dateinamenexpandierung findet immer vor irgendwelchen anderen Expandierungen statt. Sind nach dieser Expandierung eventuell noch andere Expandierungszeichen vorhanden, wird dann anschließend noch für diese die entsprechende Expandierung durchgeführt, wie z.B.

```
ls /usr/local/{*bin,lib}
```

könnte zu folgender Kommandozeile expandiert werden:

```
ls /usr/local/bin /usr/local/lib /usr/local/sbin und
```

```
ls /usr/{local/{bin,lib},include/a*.h}
```

könnte zu folgender Kommandozeile expandiert werden:

```
ls /usr/include/a*.h /usr/local/bin /usr/local/lib
```

## 7.8.2 Tilde-Expandierung

Die **bash** bietet eine Tilde-Expandierung (~) ähnlich zur C-Shell an. Die folgende Tabelle zeigt, wie Worte, die mit ~ beginnen, expandiert werden.

|                 |                                                                 |
|-----------------|-----------------------------------------------------------------|
| ~               | entspricht <b>\$HOME</b>                                        |
| ~/pfad          | entspricht <b>\$HOME/pfad</b>                                   |
| ~loginname/pfad | pfad im Home-Directory des Benutzers mit Login <i>loginname</i> |
| ~/pfad          | entspricht <b>\$PWD/pfad</b>                                    |
| ~-/pfad         | entspricht <b>\$OLDPWD/pfad</b>                                 |

Tilde-Expandierung wird auch für Worte durchgeführt, die einem Doppelpunkt (:), einem Umlenkungs-Operator oder einem = in einer Zuweisung folgen. Angabe von Pfadnamen mit ~ ist somit z.B. in **PATH**, **CDPATH** oder **MAIL-PATH** möglich.

## 7.8.3 Parameter- und Variablensubstitution

Die einfachste Zugriffsmöglichkeit auf den Wert einer Variable ist **\$variable**

Die weiteren Zugriffsmöglichkeiten auf den Wert einer Variablen entsprechen in etwa denen in der Korn-Shell:

| Variable                                              | Zugriffsmöglichkeit                                                                                                                                                                                                                                                       |
|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>\${variable}</b>                                   | liefert wie <b>\$variable</b> den Wert von <i>variable</i> . Diese Angabe muß verwendet werden, wenn dem Variablennamen ein Buchstabe, eine Ziffer oder ein Unterstrich folgt, oder wenn auf Positionsparameter mit mehr als einer Ziffer (größer als 9) zuzugreifen ist. |
| <b>\${variable:-wort}</b><br><b>\${variable-wort}</b> | Verwendung des default-Werts <i>wort</i> , wenn <i>variable</i> nicht gesetzt ist (siehe auch <b>sh</b> ).                                                                                                                                                                |
| <b>\${variable:=wort}</b><br><b>\${variable=wort}</b> | Zuweisung des default-Werts <i>wort</i> an <i>variable</i> , wenn <i>variable</i> nicht gesetzt ist (siehe auch <b>sh</b> ).                                                                                                                                              |
| <b>\${variable:?wort}</b><br><b>\${variable?wort}</b> | Fehlermeldung <i>wort</i> , wenn <i>variable</i> Nullwert hat oder nicht gesetzt ist (siehe auch <b>sh</b> ).                                                                                                                                                             |
| <b>\${variable:+wort}</b><br><b>\${variable+wort}</b> | Verwendung eines alternativen Werts (siehe auch <b>sh</b> ).                                                                                                                                                                                                              |
| <b>\${#variable}</b>                                  | (wie in <i>ksh</i> ) liefert die Länge des Strings, der in <i>variable</i> gespeichert ist. Wird für <i>variable *</i> oder <i>@</i> angegeben, wird die Anzahl der Positionsparameter geliefert.                                                                         |



| Variable                           | Zugriffsmöglichkeit                                                                                                                                                                                                                           |
|------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\${variable#pattern}</code>  | (wie in ksh) liefert den Wert von <i>variable</i> ohne den <u>kleinstmöglichen</u> durch <i>pattern</i> abgedeckten linken Teilstring oder, wenn keine Abdeckung durch <i>pattern</i> vorliegt, den vollständigen Wert von <i>variable</i> .  |
| <code>\${variable##pattern}</code> | (wie in ksh) liefert den Wert von <i>variable</i> ohne den <u>größtmöglichen</u> durch <i>pattern</i> abgedeckten linken Teilstring oder, wenn keine Abdeckung durch <i>pattern</i> vorliegt, den vollständigen Wert von <i>variable</i> .    |
| <code>\${variable%pattern}</code>  | (wie in ksh) liefert den Wert von <i>variable</i> ohne den <u>kleinstmöglichen</u> durch <i>pattern</i> abgedeckten rechten Teilstring oder, wenn keine Abdeckung durch <i>pattern</i> vorliegt, den vollständigen Wert von <i>variable</i> . |
| <code>\${variable%%pattern}</code> | (wie in ksh) liefert den Wert von <i>variable</i> ohne den <u>größtmöglichen</u> durch <i>pattern</i> abgedeckten rechten Teilstring oder, wenn keine Abdeckung durch <i>pattern</i> vorliegt, den vollständigen Wert von <i>variable</i> .   |

### 7.8.4 Kommandosubstitution

Kommandos, deren Ausgabe von der **bash** als Teil der Kommandozeile zu verwenden ist, müssen mit »Gegen-Apostrophen« (*backquotes* oder *accents graves*) geklammert werden: ``kommandos``. Alle Metazeichen behalten innerhalb einer Kommandosubstitution ihre Sonderbedeutung.

Eine Schachtelung von Kommandosubstitutionen ist mit der Angabe von ``\.`\`` (bzw. `\\\`\\\``, usw.) möglich. Zu ``cat datei`` existiert eine äquivalente, aber schnellere Variante `<datei``.

Die **bash** bietet wie die Korn-Shell noch eine andere Variante von Kommandosubstitution an: `$(kommandos)`.

Bei dieser neuen Kommandosubstitution gelten keine besonderen Quoting-Regeln. Auch bei dieser Kommandosubstitution ist Schachtelung möglich. Zu `$(cat datei)` existiert eine äquivalente, aber schnellere Angabe `$(<datei)`. Werden innerhalb von `$(.)` runde Klammern verwendet, ist deren Sonderbedeutung durch Voranstellen von `\` auszuschalten.

### 7.8.5 Arithmetische Expandierung

Um das Ergebnis eines arithmetischen Ausdrucks zu erhalten, ist entweder

`$(ausdr)` oder  
`$((ausdr))`

anzugeben. So gibt z.B. `echo ${4+5*6}` den Wert **34** aus. Der angegebene Ausdruck wird dabei so behandelt, als ob er in `"..."` angegeben wurden wäre, so daß

innerhalb des Ausdrucks Parameter- und Kommandosubstitution möglich ist. Schachtelung von arithmetischen Ausdrücken ist auch möglich. So gibt z.B. **echo \$((10+\$[4+5\*6]))** den Wert **44** aus (siehe auch arithmetische Auswertungen).

### 7.8.6 Prozeß-Substitution

Prozeß-Substitution wird auf Systemen unterstützt, die benannte Pipes (FIFOs) anbieten. Statt Pipes können zur E/A-Umlenkung auch sogenannte FIFO-Dateien verwendet werden. FIFO steht für *first-in-first-out* und realisiert eine Pipe in Form einer Datei.

```
$ mkfifo fifodatei
$ ls -l >fifodatei
$ more <fifodatei
```

Durch die drei obigen Kommandos wird zuerst die FIFO-Datei *fifodatei* eingerichtet. Anschließend wird **ls** als Hintergrundprozeß gestartet, der seine Ausgaben in die *fifodatei* schreibt. Von dort liest **more** dann die Daten wieder aus und zeigt sie seitenweise am Bildschirm an.

### 7.8.7 Dateinamenexpandierung

Die Dateinamenexpandierung der **bash** entspricht weitgehend der Dateinamenexpandierung der Bourne-Shell.

- \* steht für »eine beliebige Zeichenfolge« (auch die leere)
- ? steht für »ein beliebiges einzelnes Zeichen«
- [...] steht für »eines der in [...] angegebenen Zeichen«. Bereichsangaben sind innerhalb von [...] erlaubt.
- [!...] steht für »ein Zeichen, welches nicht in [...] angegeben ist«. Bereichsangaben sind innerhalb von [!...] erlaubt. Statt [!...] ist auch [^...] möglich.

Einige Zeichenfolgen in Dateinamen werden nur expandiert, wenn sie explizit im entsprechenden pattern angegeben sind:

.(Punkt) am Anfang eines Dateinamens  
/  
/

Ist jedoch **glob\_dot\_filenames** gesetzt, deckt **bash** bei der Dateinamenexpandierung auch Dateinamen ab, die mit .(Punkt) beginnen.

Die Dateinamenexpansion kann auch ausgeschaltet werden, indem entweder beim **bash**-Aufruf die Option **-f** angegeben wird oder aber diese Option mit **set -f** eingeschaltet wird.

Ist **allow\_null\_glob\_expansion** gesetzt, wird bei einer Dateinamenexpansion, die keinerlei Dateien abdeckt, nicht der angegebene String, sondern der Null-String eingesetzt.

## 7.9 Ein-/Ausgabeumlenkung

Die Ein-/Ausgabeumlenkung der **bash** entspricht weitgehend der der Bourne-Shell, hat aber auch einige Neuheiten aus der Korn- und C-Shell übernommen.

|                                                               |                                                                                                                                                                                                                |
|---------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&lt;datei</code>                                        | Standardeingabe in <i>datei</i> umlenken.                                                                                                                                                                      |
| <code>&gt;datei</code>                                        | (wie in ksh) Standardausgabe in <i>datei</i> umlenken.                                                                                                                                                         |
| <code>&gt; datei</code>                                       | Existiert <i>datei</i> noch nicht, wird sie neu angelegt. Ist Option <b>noclobber</b> oder <b>-C</b> gesetzt und eine existierende <i>datei</i> ist zu überschreiben, muß <code>&gt; </code> verwendet werden. |
| <code>&gt;&gt;datei</code>                                    | Standardausgabe an das Ende von <i>datei</i> umlenken.                                                                                                                                                         |
| <code>&amp;&gt; datei,</code><br><code>&gt;&amp; datei</code> | (wie in csh) Standardausgabe und Standardfehlerausgabe in <i>datei</i> umlenken. Angabe <code>&amp;&gt; datei</code> ist äquivalent zur Angabe <code>&gt;datei 2&gt;&amp;1</code> .                            |
| <code>&lt;&lt;wort</code>                                     | Hier-Dokument ( <i>here document</i> ): entspricht weitgehend dem Hier-Dokument der Bourne-Shell.                                                                                                              |
| <code>&lt;&lt;-wort</code>                                    | (wie in Bourne-Shell) identisch zu <code>&lt;&lt;wort</code> , außer daß in den nachfolgenden Eingabezeilen alle führenden Tabulatorzeichen ignoriert werden.                                                  |
| <code>&lt;&amp; fd</code>                                     | (wie in ksh) verwendet die Datei, die mit dem Filedeskriptor <i>fd</i> (Ziffer) verbunden ist, als Standardeingabe.                                                                                            |
| <code>&gt;&amp; fd</code>                                     | (wie in ksh) verwendet die Datei, die mit dem Filedeskriptor <i>fd</i> verbunden ist, als Standardausgabe.                                                                                                     |
| <code>&lt;&gt;datei</code>                                    | (wie in ksh) <i>datei</i> zum Lesen und Schreiben eröffnen.                                                                                                                                                    |
| <code>&lt;&amp;-</code>                                       | (wie in ksh) Standardeingabe schließen.                                                                                                                                                                        |
| <code>&gt;&amp;-</code>                                       | (wie in ksh) Standardausgabe schließen.                                                                                                                                                                        |

Bei all diesen Notationen kann zusätzlich vor dem entsprechenden Umlenkungszeichen noch ein Filedeskriptor (Zahl) angegeben werden, der den umzulenkenden »Datenstrom« (wie z. B. 2 für Standardfehlerausgabe) festlegt.

Für das nach einem Umlenkungsoperator angegebene Wort wird {...}-Expandierung, Tilde-Expandierung, Kommandosubstitution und arithmetische Expandierung durchgeführt. Falls daraus mehr als ein Wort resultiert, meldet die **bash** einen Fehler. Die Reihenfolge, in der die Umlenkungsoperatoren angegeben sind, ist wichtig. So lenkt z.B.

```
ls >datei 2>&1
```

sowohl die Standardausgabe als auch die Standardfehlerausgabe in *datei* um, während das Kommando

```
ls 2>&1 >datei
```

nur die Standardausgabe in *datei* umlenkt, weil *stderr* als *stdout* dupliziert wurde, bevor *stdout* in *datei* umgelenkt wurde.

## 7.10 Funktionen

Die Syntax für eine Funktionsdefinition ist dabei zunächst wie in der Bourne-Shell:

```
funktionsname() { kdoliste; }
```

Eine alternative Angabe für Funktionsdefinitionen in der **bash** ist wie in der Korn-Shell:

```
function funktionsname { kdoliste; }
```

Bei Funktionsaufrufen können Argumente angegeben werden, die dann der entsprechenden Funktion in Form von Positionsparametern zur Verfügung gestellt werden. Für die Wahl eines Funktionsnamens gelten die gleichen Regeln wie für die Wahl von Datei- und Variablennamen.

Gleiche Funktions- und Shellvariablen-Namen sind erlaubt.

Funktionen sind nur in der Shell bekannt, in der sie definiert wurden. Soll eine Funktion auch an Subshells exportiert werden, muß der entsprechende Funktionsname mit **export -f** in der *Environment*-Datei definiert werden.

In **bash** ist es möglich, funktionslokale Variablen mit dem builtin-Kommando **local** zu deklarieren.

**bash**-Funktionen können auch rekursiv aufgerufen werden.

Ein-/Ausgabeumlenkung ist sowohl bei der Definition als auch beim Aufruf einer Shell-Funktion erlaubt.

**return** bewirkt das Verlassen einer Funktion.

Die in der gerade aktiven **bash** definierten Funktionen können mit **typeset -f** oder mit **declare -f** angezeigt werden.

Mit **unset -f *funktionsname*** kann die Definition einer Funktion wieder gelöscht werden.

Da ein builtin-Kommando durch eine Funktion ersetzt werden kann, kann innerhalb der Funktion das entsprechende builtin-Kommando mittels des neuen Kommandos **builtin** aufgerufen werden.

## 7.11 Alias-Mechanismus

Der Alias-Mechanismus der **bash** entspricht in etwa dem Alias-Mechanismus der **ksh**, nur daß das **alias**-Kommando der **bash** die beiden Optionen **-t** und **-x** nicht kennt. Beim Kommando **unalias**, das das Löschen von Aliase erlaubt, ist die Option **-a** möglich, um alle Aliase auf einmal zu löschen.

### **alias**

gibt alle momentan definierten Aliase aus.

### **alias *name***

gibt den Wert des Alias *name* aus, wenn ein solches existiert.

### **alias *name*=wert**

definiert ein Alias *name* mit dem angegebenen *wert*. *wert* kann ein beliebiger Text sein. Für *wert* wird Parameter-, Kommandosubstitution und Dateinamenexpansion durchgeführt.

Handelt es sich beim ersten Wort eines einfachen Kommandos um ein Alias, dann setzt die **bash** hierfür den entsprechenden Alias-Wert ein. Kommt im Alias-Text der Alias-Name selbst nochmals vor, wird dieser nicht erneut substituiert. Kommt im Alias-Text dagegen ein anderer Alias-Name vor, wird auch dieser substituiert.

Nur wenn ein Alias-Text mit einem Leer- oder Tabulatorzeichen endet, wird auch noch für das nachfolgende Wort Alias-Substitution durchgeführt.

## 7.12 Jobkontrolle

Der Job, der gerade im Vordergrund abgearbeitet wird, kann mit der *Suspend*-Taste (**Strg-z**) angehalten werden. Für die Jobkontrolle stehen die builtin-Kommandos **jobs**, **kill**, **bg**, **fg** und **wait** zur Verfügung. Die einzelnen Jobs können über die PID, die Jobnummer oder den Jobnamen angesprochen werden. Um einen Job über seine Jobnummer oder seinen Namen anzusprechen, gibt es folgende Notationen:

| Notation                | Beschreibung                               |
|-------------------------|--------------------------------------------|
| <code>%jobnr</code>     | Job mit Jobnummer <i>jobnr</i>             |
| <code>%string</code>    | Job, dessen Name mit <i>string</i> beginnt |
| <code>/?string</code>   | Job, dessen Name <i>string</i> enthält     |
| <code>%+ oder %%</code> | momentaner Job                             |
| <code>%-</code>         | vorheriger aktueller Job                   |

Die Angabe `%jobnr` ist identisch zu `fg %jobnr` und die Angabe `%jobnr &` ist identisch zu `bg %jobnr`.

## 7.13 Der bash-Prompt

Jedesmal, bevor die **bash** den Primär-Prompt **PS1** ausgibt, prüft sie den Inhalt der Variablen **PROMPT\_COMMAND**. Ist diese Variable gesetzt, führt sie deren Inhalt aus, wie wenn man die dort angegebenen Kommandos direkt eingegeben hätte.

In der Variable **PS1** können die folgenden Sonderzeichen angegeben werden:

| Sonderzeichen     | Beschreibung                                       |
|-------------------|----------------------------------------------------|
| <code>\t</code>   | Zeit im Format HH:MM:SS                            |
| <code>\d</code>   | Datum im Format »Wochentag Monat Tag« (Thu Dec 14) |
| <code>\n</code>   | Neuzeile-Zeichen                                   |
| <code>\s</code>   | Name der Shell (Basisname von <b>\$0</b> )         |
| <code>\w</code>   | Aktuelles Working-Directory                        |
| <code>\W</code>   | Basisname von <b>\$PWD</b>                         |
| <code>\u</code>   | Benutzername                                       |
| <code>\h</code>   | Hostname                                           |
| <code>\#</code>   | Nummer des aktuellen Kommandos                     |
| <code>!\</code>   | History-Nummer des aktuellen Kommandos             |
| <code>\nnn</code> | Zeichen mit oktalem Ascii-Code <i>nnn</i>          |
| <code>\\$</code>  | wenn effektive UID 0 ist, dann #, sonst \$         |
| <code>\\</code>   | Backslash                                          |
| <code>\[</code>   | Beginn für eine Reihe von nichtdruckbaren Zeichen  |
| <code>\]</code>   | Ende einer Reihe von nichtdruckbaren Zeichen       |

## 7.14 Editieren von Kommandozeilen

Das Editieren von Kommandozeilen ist möglich, wenn die Option **-nolineediting** nicht gesetzt ist. Die Editor-Kommandos entsprechen weitgehend denen von **emacs**. Ein **vi**-ähnlicher Editor steht auch zur Verfügung.

In diesem Abschnitt wird die emacs-Notation benutzt: Control-Tasten werden mit **C-**, Meta-Tasten werden mit **M-** angegeben. So bedeutet z.B. die Angabe **C-x**, daß die Taste *x* bei gedrückter *Strg*-Taste zu drücken ist. Auf Tastaturen ohne Meta-Taste (*Alt*-Taste) entspricht **M-** dem Drücken der *ESC*-Taste. So bedeutet z.B. die Angabe **M-w**, daß zunächst die *ESC*-Taste und dann die Taste *w* zu drücken ist. **M-C-x** bedeutet dann, daß zunächst die *ESC*-Taste und dann die Taste *x* bei gedrückter *Strg*-Taste einzugeben ist.

Die voreingestellten Aktionen für Tasten, die sogenannten *key-bindings*, kann der Benutzer über entsprechende Angaben in der Datei `~/inputrc` ändern. Wenn die Shellvariable **INPUTRC** gesetzt ist, wird anstelle von `~/inputrc` die dort enthaltene Datei verwendet. Trägt z.B. der Benutzer eine der beiden folgenden Zeile in `~/inputrc` ein:

**M-Control-r:** clear-screen oder  
**C-Meta-r:** clear-screen

so wird bei Eingabe von **M-C-r** der Bildschirm gelöscht.

Die symbolischen Zeichennamen **RUBOUT**, **DEL**, **ESC**, **LFD**, **NEWLINE**, **RET**, **RETURN**, **SPC**, **SPACE** und **TAB** werden als Schlüsselwörter erkannt.

In `~/inputrc` sind nur einige wenige Konstrukte erlaubt:

- ▶ Leerzeilen werden ignoriert.
- ▶ Zeilen, die mit **#** beginnen, sind Kommentar.
- ▶ Zeilen, die mit **\$** beginnen, sind Direktiven.
- ▶ Alle anderen Zeilen sind *key-bindings* oder Zuweisungen an Variablen.

Die Syntax für *key-bindings* in `~/inputrc` ist sehr einfach. Es ist lediglich die Tastenkombination und der Name des Kommandos oder Makros anzugeben, das für diese Tastenkombination auszuführen ist. Die Tastenkombination kann auf zwei verschiedene Arten angegeben werden: als symbolischer Tastenname (A, B, ..), dem möglicherweise ein Meta- oder Control- vorangestellt ist, wie z.B.

**Control-r:** forward-char  
**Meta-Rubout:** backward-kill-word  
**Control-O:** ">&/tmp/bloedsinn"

Im obigen Beispiel wird **C-r** auf *forward-char* (Cursor ein Zeichen nach rechts), **M-DEL** auf *backward-kill-word* (ein Wort links löschen) und **C-o** auf das Einfügen des Textes `>&/tmp/bloedsinn` festgelegt.

Eine zweite Möglichkeit des key-bindings ist die Angabe einer Folge von Tastenkombinationen (in ".." angegeben) anstelle von lediglich einer Tastenkombination, wie z.B.

**"\C-r": forward-char**

**"\C-x\C-r": re-read-init-file**

**"\e[11~": "Funktionstaste F1"**

In diesem Beispiel wird *C-r* wieder auf *forward-char* (Cursor ein Zeichen nach rechts) festgelegt. *C-x C-r* wird auf **re-read-init-file** (erneutes Lesen von *~/inputrc*) und *ESC [ 1 1 ~* wird auf das Einfügen des Textes *Funktionstaste F1* festgelegt.

Folgende Escape-Sequenzen sind dabei möglich:

**\C-** Control-Präfix

**\M-** Meta-Präfix

**\e-** ESC

**\ \-** Backslash

**\ "-** Anführungszeichen "

**\ '-** Hochkomma '

Wenn der Text eines Makros eingegeben wird, sollte dieser in "..." oder in '...' angegeben werden, sonst wird angenommen, daß es sich bei diesem Text um einen Funktionsnamen handelt. Um die Sonderbedeutung eines Zeichens im Makrotext auszuschalten, ist diesem ein Backslash \ voranzustellen.

Das Modifizieren oder Anzeigen der aktuellen key-bindings ist mit dem builtin-Kommando **bind** möglich. Umschalten oder Ausschalten eines Editors ist mit **set -o emacs** oder **set -o vi** möglich.

Um in *~/inputrc* eine Variable zu setzen, ist

**set variablename wert** (wie z.B. **set editing-mode vi**)

anzugeben.

Nachfolgend sind die Variablen angegeben, die in *~/inputrc* gesetzt werden können. Der Default-Wert ist dabei in Klammern angegeben:

**horizontal-scroll-mode** (Off)

Wenn auf *On* gesetzt, wird bei überlangen Zeilen kein automatischer Zeilenumbruch (für die Anzeige) durchgeführt, sondern der überlange Teil der Zeile angezeigt, wobei die vorderen Zeichen dieser Zeile erst wieder sichtbar werden, wenn man den Cursor dorthin bewegt.

**editing-mode** (emacs)

legt fest, ob key-bindings für *emacs* oder *vi* durchgeführt werden.



**mark-modified-lines** (Off)

Wenn auf *On* gesetzt, werden History-Zeilen, die modifiziert wurden, durch einen vorangestellten \* gekennzeichnet.

**bell-style** (audible)

Wenn auf *none* gesetzt, wird niemals ein akustisches Signal beim Editieren von Zeilen ausgegeben. Ist diese Variable auf *visible* gesetzt, so wird visuelles Signal (wenn verfügbar) bei entsprechenden Aktionen angezeigt. Ist *bell-style* auf *audible* gesetzt, wird bei entsprechenden Aktionen ein akustisches Signal ausgegeben.

**comment-begin** (#)

definiert den String, der im vi-Modus eingefügt wird, wenn das vi-Kommentar-kommando (**vi-comment**) ausgeführt wird.

**meta-flag** (Off)

Wenn auf *On* gesetzt, wird 8-Bit-Eingabe eingeschaltet.

**convert-meta** (On)

Wenn auf *On* gesetzt, werden Zeichen, bei denen das 8. Bit gesetzt ist, in ein 7-Bit-Ascii-Zeichen umgewandelt, dem ein ESC vorangestellt wird.

**output-meta** (Off)

Wenn auf *On* gesetzt, werden 8-Bit-Zeichen direkt ohne vorangestelltes ESC ausgegeben.

**completion-query-items** (100)

legt Anzahl fest, ab der der Benutzer gefragt wird, ob er die durch Wortvervollständigung entstandenen Namen wirklich sehen möchte.

**keymap** (emacs)

legt die momentanen key-bindings fest. Mögliche Werte hierbei sind **emacs**, **emacs-standard**, **emacs-meta**, **emacs-ctlx**, **vi**, **vi-move**, **vi-command** und **vi-insert**. **vi** ist äquivalent zu **vi-command** und **emacs** ist äquivalent zu **emacs-command**.

**show-all-if-ambiguous** (Off)

Wenn auf *On* gesetzt, wird bei Wörtern, die mehrfache Vervollständigungen zulassen, sofort die entsprechende Liste von möglichen Wörtern ausgegeben, und nicht erst die Terminalglocke geläutet.

**expand-tilde** (Off)

Wenn auf *On* gesetzt, wird Tilde-Expandierung bei Wort-Vervollständigung durchgeführt.

In *~/inputrc* können Konstrukte angegeben werden, die ähnlich zur bedingten Kompilierung in C sind.

**\$if**

Mit dem **\$if**-Konstrukt ist es möglich, abhängig vom Editier-Modus, dem benutzten Terminal oder einer bestimmten Anwendung, Variablen zu setzen oder key-bindings durchzuführen.

**mode**

Mit **mode=emacs** kann überprüft werden, ob emacs-Modus und mit **mode=vi**, ob vi-Modus eingestellt ist.

**term**

Mit **term=terminalname** kann überprüft werden, ob ein bestimmtes Terminal benutzt wird. Abhängig davon können dann bestimmte key-bindings für Funktionstasten durchgeführt werden.

**application**

Mit der Angabe eines Namens *name* für *application* kann überprüft werden, ob ein bestimmtes Anwendungsprogramm mit Namen *name* gerade abläuft. So können programmspezifische key-bindings durchgeführt werden. z.B. wird durch folgende Angabe das Einschließen des aktuellen und des vorherigen Worts in "..." für die Tastenkombination *Strg-xq* festgelegt. Diese Festlegung wird jedoch nur getroffen, wenn momentan die **bash** aktiv ist.

```
$if bash
Umgeben des aktuellen und vorherigen Worts mit "..."
"\C-xq": "\eb"\\ef\"
$endif
```

**\$endif**

schließt einen **\$if**-Konstrukt ab.

**\$else**

leitet einen else-Zweig im **\$if**-Konstrukt ein.

Editier-Kommandos können numerische Argumente (positive oder negative Zahlen) übergeben werden, indem vor der Eingabe des Kommandos **M-n** (wie z.B. *ESC4*) eingegeben wird. Meistens legen diese Argumente fest, wie oft das entsprechende Kommando auszuführen ist. Manchmal ist aber auch das Vorzeichen der angegebenen Zahl wichtig. Wenn nämlich einem Kommando, das normalerweise nach rechts arbeitet, eine negative Zahl als Argument übergeben wird, arbeitet es in die umgekehrte Richtung, nämlich nach links. In der nachfolgenden Liste wird dieses Verhalten bei entsprechenden Kommandos explizit hervorgehoben.

Kommandos, die Text löschen, heben den gelöschten Text in einem Ringpuffer auf.

Nachfolgend wird eine Liste der möglichen Editier-Kommandonamen gegeben, wobei die voreingestellten Tastenkombinationen (key-bindings) in Klammern angegeben sind.

| <i>Cursorpositionierung</i>                         |                                                                                                                                                                        |
|-----------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>beginning-of-line (C-a)</b>                      | an Zeilenanfang                                                                                                                                                        |
| <b>end-of-line (C-e)</b>                            | ans Zeilenende                                                                                                                                                         |
| <b>forward-char (C-f)</b>                           | ein Zeichen nach rechts                                                                                                                                                |
| <b>backward-char (C-b)</b>                          | ein Zeichen nach links                                                                                                                                                 |
| <b>forward-word (M-f)</b>                           | Ein Wort nach rechts                                                                                                                                                   |
| <b>backward-word (M-b)</b>                          | Ein Wort nach links                                                                                                                                                    |
| <b>clear-screen (C-l)</b>                           | Bildschirm löschen und aktuelle Zeile oben anzeigen. Ist ein Argument angegeben, wird nur aktuelle Zeile gelöscht und erneut angezeigt.                                |
| <b>redraw-current-line</b>                          | aktuelle Zeile löschen und erneut anzeigen                                                                                                                             |
| <i>Editieren von History-Kommandos</i>              |                                                                                                                                                                        |
| <b>accept-line (Return)</b>                         | Vollständige aktuelle Zeile (unabhängig von Cursorposition) zur Ausführung bringen. Falls Zeile nicht leer ist, wird sie in der History-Liste aufgenommen.             |
| <b>previous-history (C-p)</b>                       | Vorherige Kommandozeile aus History-Liste holen                                                                                                                        |
| <b>next-history (C-n)</b>                           | Nächste Kommandozeile aus History-Liste holen                                                                                                                          |
| <b>beginning-of-history (M-<b>&lt;</b>)</b>         | Auf älteste (erste) Zeile in History-Liste positionieren                                                                                                               |
| <b>end-of-history (M-<b>&gt;</b>)</b>               | Auf letzte (aktuelle) Zeile in History-Liste positionieren                                                                                                             |
| <b>reverse-search-history (C-r)</b>                 | Rückwärts in History-Liste suchen. Bei jedem nach <b>C-r</b> eingegebenen Zeichen wird sofort die Zeile angezeigt, in der der bisher eingegebene String enthalten ist. |
| <b>forward-search-history (C-s)</b>                 | Vorwärts in History-Liste suchen. Bei jedem nach <b>C-s</b> eingegebenen Zeichen wird sofort die Zeile angezeigt, in der der bisher eingegebene String enthalten ist.  |
| <b>non-incremental-reverse-search-history (M-p)</b> | Rückwärts in History-Liste suchen. Nach dem nach <b>M-p</b> eingegebenen String wird erst gesucht, wenn Return eingegeben wird.                                        |
| <b>non-incremental-forward-search-history (M-n)</b> | Vorwärts in History-Liste suchen. Nach dem nach <b>M-n</b> eingegebenen String wird erst gesucht, wenn Return eingegeben wird.                                         |
| <b>history-search-forward</b>                       | Vorwärts in History-Liste nach dem String suchen, der sich zwischen Zeilenanfang und aktueller Cursorposition befindet.                                                |

|                                         |                                                                                                                                                                                                                                                                                             |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>history-search-backward</b>          | Rückwärts in History-Liste nach dem String suchen, der sich zwischen Zeilenanfang und aktueller Cursorposition befindet.                                                                                                                                                                    |
| <b>yank-nth-arg (M-C-y)</b>             | An Cursorposition das 1. Argument (2. Wort) der vorherigen Kommandozeile einfügen. Ist ein Argument <i>n</i> angegeben, wird das <i>n</i> .te Wort aus der vorherigen Kommandozeile eingefügt. Ein negatives <i>n</i> fügt das <i>n</i> .te Wort vom Ende der vorherigen Kommandozeile ein. |
| <b>yank-last-arg (M-., M-_)</b>         | An Cursorposition das letzte Argument (letztes Wort) der vorherigen Kommandozeile einfügen. Ist ein Argument angegeben, verhält es sich genau wie <b>yank-nth-arg</b> .                                                                                                                     |
| <b>shell-expand-line (M-C-e)</b>        | Alias-, History- und Wort-Expandierung für Kommandozeile durchführen, wie es die <b>bash</b> tut.                                                                                                                                                                                           |
| <b>history-expand-line (M-^)</b>        | History-Expandierung für Kommandozeile durchführen.                                                                                                                                                                                                                                         |
| <b>insert-last-argument (M-., M-_)</b>  | entspricht <b>yank-last-arg</b> .                                                                                                                                                                                                                                                           |
| <b>operate-and-get-next (C-o)</b>       | Aktuelle Zeile an <b>bash</b> zur Verarbeitung übergeben und nächste History-Zeile holen; wird für das Editieren mehrzeiliger Kommandos benötigt.                                                                                                                                           |
| <i>Ändern von Text</i>                  |                                                                                                                                                                                                                                                                                             |
| <b>delete-char (C-d)</b>                | Zeichen an Cursorposition löschen (Am Anfang einer Leerzeile entspricht es EOF)                                                                                                                                                                                                             |
| <b>backward-delete-char (Rubout)</b>    | Zeichen links vom Cursor löschen. Ist ein Argument angegeben, wird gelöschter Text im Ringpuffer gesichert.                                                                                                                                                                                 |
| <b>quoted-insert (C-q, C-v)</b>         | Ausschalten des nachfolgenden Sonderzeichens                                                                                                                                                                                                                                                |
| <b>tab-insert (C-v TAB)</b>             | Einfügen eines Tabzeichens                                                                                                                                                                                                                                                                  |
| <b>self-insert (a ,b, A, 1, !, ...)</b> | Einfügen des entsprechenden Zeichens                                                                                                                                                                                                                                                        |
| <b>transpose-chars (C-t)</b>            | Zeichen an Cursorposition mit vorherigem vertauschen. Negative Argumente haben hier keine Auswirkung.                                                                                                                                                                                       |
| <b>transpose-words (M-t)</b>            | Aktuelles Cursorwort mit vorherigem Wort vertauschen                                                                                                                                                                                                                                        |
| <b>upcase-word (M-u)</b>                | Aktuelles (oder folgendes Wort) in Großbuchstaben umwandeln. Bei negativem Argument wird vorheriges Wort (ohne Cursorbewegung) in Großbuchstaben umgewandelt.                                                                                                                               |

|                                            |                                                                                                                                                                                                                   |
|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>downcase-word (M-l)</b>                 | Aktuelles (oder folgendes Wort) in Kleinbuchstaben umwandeln. Bei negativem Argument wird vorheriges Wort (ohne Cursorbewegung) in Kleinbuchstaben umgewandelt.                                                   |
| <b>capitalize-word (M-c)</b>               | Bei aktuellem (oder folgendem) Wort Anfangsbuchstaben in Großbuchstaben und Rest in Kleinbuchstaben umwandeln. Bei negativem Argument wird diese Umwandlung auf vorheriges Wort (ohne Cursorbewegung) angewendet. |
| <i>Löschen und Sichern im Puffer</i>       |                                                                                                                                                                                                                   |
| <b>kill-line (C-k)</b>                     | Ab Cursor bis Zeilenende löschen                                                                                                                                                                                  |
| <b>backward-kill-line (C-x Rubout)</b>     | Ab Cursor bis Zeilenanfang löschen                                                                                                                                                                                |
| <b>unix-line-discard (C-u)</b>             | Ab Cursor bis Zeilenanfang löschen                                                                                                                                                                                |
| <b>kill-whole-line</b>                     | Ganze Zeile löschen                                                                                                                                                                                               |
| <b>kill-word (M-d)</b>                     | Ab Cursor bis Wortende löschen                                                                                                                                                                                    |
| <b>backward-kill-word (M-Rubout)</b>       | Ab Cursor bis Wortanfang löschen                                                                                                                                                                                  |
| <b>unix-word-rubout (C-w)</b>              | Ab Cursor bis Wortanfang löschen                                                                                                                                                                                  |
| <b>delete-horizontal-space</b>             | Alle Leer- und Tabzeichen um den Cursor löschen                                                                                                                                                                   |
| <b>yank (C-y)</b>                          | Obersten Text des Ringpuffers an Cursorposition einfügen                                                                                                                                                          |
| <b>yank-pop (M-y)</b>                      | Ringpuffer rotieren und neuen obersten Text des Ringpuffers an Cursorposition einfügen. Funktionierte nur, wenn dieses Kommando direkt einem <b>yank</b> oder <b>yank-pop</b> folgt.                              |
| <i>Numerische Argumente</i>                |                                                                                                                                                                                                                   |
| <b>digit-argument (M-0, M-1, ..., M--)</b> | M-- leitet ein negatives Argument ein.                                                                                                                                                                            |
| <b>universal-argument</b>                  | Jedesmal, wenn dieses Kommando ausgeführt wird, wird der Argument-Zähler mit vier multipliziert. Der Argument-Zähler ist ursprünglich mit eins initialisiert.                                                     |

| <i>Automatisches Vervollständigen</i>         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>complete (TAB)</b>                         | <b>bash</b> versucht eine Vervollständigung für aktuelles Wort (Text vor Cursor). Beginnt Wort mit \$, versucht sie einen existierenden Variablennamen hieraus zu machen. Beginnt Wort mit ~, versucht sie einen existierenden Benutzernamen hieraus zu machen. Beginnt Wort mit @, versucht sie einen existierenden Hostnamen hieraus zu machen. Trifft keine dieser vorherigen Bedingungen zu, versucht sie hieraus einen existierenden Alias- oder Funktionsnamen zu machen. Sollte auch dies fehlschlagen, versucht sie einen existierenden Dateinamen aus dem Wort zu machen. |
| <b>possible-completions (M-?)</b>             | Auflisten aller möglichen Vervollständigungen für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>insert-completions</b>                     | Einfügen aller möglichen Vervollständigungen für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>complete-filename (M-/)</b>                | Dateinamen-Vervollständigung für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>possible-filename-completions (C-x /)</b>  | Auflisten der möglichen Dateinamen für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>complete-username (M-~)</b>                | Benutzernamen-Vervollständigung für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>possible-username-completions (C-x ~)</b>  | Auflisten der möglichen Benutzernamen für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>complete-variable (M-\$)</b>               | Variablennamen-Vervollständigung für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>possible-variable-completions (C-x \$)</b> | Auflisten der möglichen Variablennamen für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>complete-hostname (M-@)</b>                | Hostnamen-Vervollständigung für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>possible-hostname-completions (C-x @)</b>  | Auflisten der möglichen Hostnamen für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>complete-command (M-!)</b>                 | Kommandonamen-Vervollständigung für Wort: Unter Kommandonamen fallen dabei Aliase, reservierte Wörter, Shell-Funktionen, builtin-Kommandos und ausführbare Dateien                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>possible-command-completions (C-x !)</b>   | Auflisten der möglichen Kommandos für Wort                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>dynamic-complete-history (M-TAB)</b>       | History-Vervollständigung: Wort wird mit den History-Zeilen verglichen                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

|                                             |                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>complete-into-braces (M-{})</b>          | Dateinamen-Vervollständigung für Wort, wobei die entsprechenden Dateinamen in einer {}-Liste angegeben werden. Wenn z. B. im aktuellen Directory die Dateien <i>sscs_rcs</i> , <i>schul</i> , <i>src</i> , <i>sysprog</i> und <i>sysprog2</i> existieren, dann würde aus dem Drücken von <b>M-{} </b> nach der Eingabe von <b>echo s</b> folgende Zeile resultieren:<br><br><b>echo "s{c{cs_rcs,hul},rc,ysprog{,2}"</b> |
| <i>Makros</i>                               |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>start-kbd-macro (C-x ()</b>              | Start der Aufzeichnung eines Makros. Ab diesen Zeitpunkt werden alle eingegebenen Zeichen im aktuellen Tastatur-Makro festgehalten.                                                                                                                                                                                                                                                                                     |
| <b>end-kbd-macro (C-x ))</b>                | Ende der Aufzeichnung eines Makros.                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>call-last-kbd-macro (C-x e)</b>          | Aufruf des letzten aufgezeichneten Tastatur-Makros: Alle dort aufgezeichneten Zeichen werden nun eingefügt/ausgeführt, als ob man sie direkt auf der Tastatur eingegeben hätte.                                                                                                                                                                                                                                         |
| <i>Sonstiges</i>                            |                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>re-read-init-file (C-x C-r)</b>          | Inhalt der Initialisierungs-Datei <i>~/inputrc</i> erneut lesen und die dort enthaltenen key-bindings oder Variablen-Zuweisungen gültig machen.                                                                                                                                                                                                                                                                         |
| <b>abort (C-g)</b>                          | aktuelles Editor-Kommando abbrechen und Terminalglocke läuten.                                                                                                                                                                                                                                                                                                                                                          |
| <b>do-uppercase-version (M-a, M-b, ...)</b> | Kommando ausführen, das entsprechenden Großbuchstaben zugeordnet ist.                                                                                                                                                                                                                                                                                                                                                   |
| <b>prefix-meta (ESC)</b>                    | Metazeichen für Editor-Kommandos. So ist z. B. <i>ESC-f</i> identisch zu <i>Alt-f</i> .                                                                                                                                                                                                                                                                                                                                 |
| <b>undo (C-_, C-x C-u)</b>                  | Letztes Änderungs-Kommando rückgängig machen. Erneutes <b>undo</b> macht vorheriges Änderungs-Kommando rückgängig, usw.                                                                                                                                                                                                                                                                                                 |
| <b>revert-line (M-r)</b>                    | Alle Änderungen für aktuelle Zeile rückgängig machen                                                                                                                                                                                                                                                                                                                                                                    |
| <b>tilde-expand (M-~)</b>                   | Tilde-Expandierung für aktuelles Wort durchführen                                                                                                                                                                                                                                                                                                                                                                       |
| <b>dump-functions</b>                       | All Funktionen und deren key-bindings auf stdout ausgeben. Ist ein numerisches Argument gegeben, wird die Ausgabe so aufbereitet, daß sie in eine <i>inputrc</i> -Datei untergebracht werden kann.                                                                                                                                                                                                                      |
| <b>display-shell-version (C-x C-v)</b>      | Ausgeben von Versions-Information zur verwendeten <b>bash</b>                                                                                                                                                                                                                                                                                                                                                           |
| <b>emacs-editing-mode (C-e)</b>             | Vom vi-Editiermodus in den emacs-Editiermodus schalten; identisch zu <b>set -o emacs</b>                                                                                                                                                                                                                                                                                                                                |

### vi-Modus

Ist der vi-Modus eingeschaltet, ist anders als beim wirklichen **vi** immer automatisch der Einfügemodus eingeschaltet, als ob man das vi-Kommando **i** (Einfügen) eingegeben hätte.

Durch Drücken von **ESC** wird im vi-Modus dann in den Kommandomodus umgeschaltet. Im Kommandomodus können dann die vom wirklichen vi her bekannten Kommandos zum Positionieren des Cursors verwendet werden. Ein Holen der vorherigen Kommandozeile aus History-Liste ist mit **k** und das Holen der nachfolgenden History-Kommandozeile mit **j** möglich.

## 7.15 History-Mechanismus

Der History-Mechanismus der **bash** ist dem History-Mechanismus der C-Shell ähnlich. Um sich die Kommandos aus der History-Liste anzeigen zu lassen, steht das builtin-Kommando **history** zur Verfügung. Wie viele Kommandos in der History-Liste aufzuheben sind, wird über die Variable **HISTSIZE** (Voreinstellung: 500) festgelegt. Die Datei, in der History-Liste zu speichern ist, wird über die Variable **HISTFILE** (Voreinstellung: `~/.bash_history`) festgelegt.

Soll auf eine zuvor eingegebene Kommandozeile wieder zugegriffen werden, muß das Ausrufezeichen **!** angegeben werden.

Dabei kann nun folgendes angegeben werden:

| Angabe             | Aus History-Liste wird in aktuelle Kommandozeile eingesetzt                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>!n</b>          | Kommandozeile mit der event number <i>n</i> .                                                                                                                                  |
| <b>!-n</b>         | <i>n</i> .te vorherige Kommandozeile.                                                                                                                                          |
| <b>!!</b>          | Unmittelbar zuvor eingegebene Kommandozeile; !! entspricht also <b>!-1</b> .                                                                                                   |
| <b>!string</b>     | Letzte Kommandozeile, die mit <i>string</i> beginnt.                                                                                                                           |
| <b>!?string[?]</b> | Letzte Kommandozeile, die <i>string</i> enthält.                                                                                                                               |
| <b>!#</b>          | Bisher eingegebene aktuelle Zeile. Mit dieser Aufrufform ist es möglich, auf zuvor (weiter links) eingegebene Wörter wieder zuzugreifen, ohne diese erneut eingeben zu müssen. |

**!** verliert seine Sonderbedeutung, wenn vor ihm ein **\** steht oder ihm ein Leer-, Tab-, Neuezeile-Zeichen, **=** oder **(** folgt.

Die spezielle Angabe **^alt^neu^** ersetzt in der vorherigen Kommandozeile den durch **alt** abgedeckten Text durch den Text *neu*. Diese Angabe ist somit äquivalent zu **!!:s/alt/neu/**.



Es ist auch möglich, daß man nicht die ganze Kommandozeile, sondern nur Teile davon einsetzen läßt:

`!angabe[:auswahl][:modifikator(en)]`

Für *angabe* muß eine der sechs zuvor beschriebenen Formen verwendet werden und für *auswahl* kann folgendes angegeben werden:

| Auswahl    | Wortauswahl in entsprechender Kommandozeile                                                               |
|------------|-----------------------------------------------------------------------------------------------------------|
| <i>n</i>   | <i>n</i> .tes Wort; <b>0</b> ist das erste Wort, <b>1</b> das zweite Wort, <b>2</b> das dritte Wort, usw. |
| <b>^</b>   | erstes Argument (nicht Kommandoname)                                                                      |
| <b>\$</b>  | letztes Argument                                                                                          |
| <b>%</b>   | das durch eine zuvor angegebene Konstruktion <i>?string?</i> ausgewählte Wort                             |
| <i>m-n</i> | <i>m</i> bis <i>n</i> .tes Wort; Wortzählung beginnt wieder bei 0                                         |
| <i>-n</i>  | entspricht <b>0-n</b>                                                                                     |
| <i>m-</i>  | <i>m</i> bis vorletztes Wort                                                                              |
| <b>*</b>   | alle Wörter außer 0.Wort; entspricht <b>^-\$</b>                                                          |
| <i>n*</i>  | entspricht <i>n-\$</i>                                                                                    |

Der Doppelpunkt : vor *auswahl* kann weggelassen werden, wenn *auswahl* mit **^**, **\$**, **\***, **-** oder **%** beginnt.

Die über die Angabe von *auswahl* ausgewählten Wörter könnten durch Angabe von *modifikatoren* noch modifiziert werden:

| Modifikator       | Auswirkung                                                                                                                                                                                                                                                                                                    |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>h</b>          | ( <b>head</b> ) liefert den Pfadnamen des Worts.                                                                                                                                                                                                                                                              |
| <b>t</b>          | ( <b>tail</b> ) liefert den Basisnamen des Worts (Gegenstück zu <b>h</b> ).                                                                                                                                                                                                                                   |
| <b>r</b>          | ( <b>remove</b> ) schneidet vom Wort eine vorhandene Endung <i>.xxx</i> ab.                                                                                                                                                                                                                                   |
| <b>e</b>          | ( <b>extension</b> ) liefert vom Wort nur eine eventuell vorhandene Endung <i>.xxx</i> .                                                                                                                                                                                                                      |
| <i>s/alt/neu/</i> | ( <b>substitute</b> ) ersetzt den durch <i>alt</i> abgedeckten Text durch <i>neu</i> . Für <i>alt</i> dürfen die bei <b>ed</b> zugelassenen regulären Ausdrücke verwendet werden.<br><br>Wird das Zeichen <b>&amp;</b> in <i>neu</i> angegeben, wird hierfür der durch <i>alt</i> abgedeckte Text eingesetzt. |

| Modifikator  | Auswirkung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | <p>Wird für <i>neu</i> keinerlei Text angegeben, wird dafür das <i>neu</i> aus einem vorherigen <i>s/alt/neu/</i> oder aus einer vorherigen Angabe <i>!string?</i> genommen.</p> <p>Statt / darf jedes beliebige Trennzeichen verwendet werden, solange es nicht in <i>alt</i> oder <i>neu</i> vorkommt.</p> <p>Wenn nach <i>s/alt/neu/</i> sofort ein Neuezeile-Zeichen angegeben wird, darf das letzte / auch weggelassen werden.</p> <p>Um die Bedeutung von Sonderzeichen wie <b>&amp;</b> oder / in <i>alt</i> oder <i>neu</i> auszuschalten, muß diesen ein \ vorangestellt werden.</p> |
| <b>&amp;</b> | wiederholt die letzte mit <i>s/alt/neu/</i> angegebene Substitution.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>g</b>     | ( <i>global</i> ) Modifikator wird nicht nur auf das erste Wort, sondern auf alle ausgewählten Wörter angewendet. <b>g</b> ist dabei vor dem Modifikator ( <b>s</b> , <b>&amp;</b> ) anzugeben.                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>p</b>     | ( <i>print</i> ) die neu entstandene Kommandozeile wird nur angezeigt, aber nicht ausgeführt.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>q</b>     | ( <i>quote</i> ) für die Kommandozeile wird Quoting vorgenommen, so daß keine weiteren Substitutionen oder Dateinamenexpandierungen stattfinden.                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>x</b>     | wie <b>q</b> , nur daß der entsprechende Wert in mehrere Worte aufgeteilt wird. Trennzeichen sind dabei Leer-, Tabulator- und Neuezeile-Zeichen.                                                                                                                                                                                                                                                                                                                                                                                                                                              |

Nur wenn **g** als erster Modifikator angegeben ist, werden die Modifikationen für alle Wörter durchgeführt, ansonsten werden sie nur für das erste Wort vorgenommen.

## 7.16 bash-Arithmetik

Die **bash** verfügt über eine eigene Arithmetik und verwendet dabei **long**-Darstellung. Sie prüft dabei niemals auf einen eventuellen *Overflow* ab. Division durch 0 wird durch Signal abgefangen und als Fehler gewertet.

Die **bash** erlaubt die direkte Angabe von arithmetischen Ausdrücken oder aber unter Verwendung des builtin-Kommandos **let**.

Die folgende Liste zeigt die erlaubten Operatoren in absteigender Priorität. Operatoren mit gleicher Priorität sind dabei zu Gruppen zusammengefaßt:

|                                                                          |                                                         |
|--------------------------------------------------------------------------|---------------------------------------------------------|
| <code>-, +</code>                                                        | Minus, Plus als Vorzeichen                              |
| <code>!, ~</code>                                                        | Logische Negation, Bitweise Negation (Einer-Komplement) |
| <code>*, /, %</code>                                                     | Multiplikation, Division, Modulo                        |
| <code>+, -</code>                                                        | Addition, Subtraktion                                   |
| <code>&lt;&lt;, &gt;&gt;</code>                                          | Links-, Rechtsshift                                     |
| <code>&lt;=, &gt;=, &lt;, &gt;</code>                                    | Vergleiche                                              |
| <code>==, !=</code>                                                      | Vergleich auf Gleichheit, Ungleichheit                  |
| <code>&amp;</code>                                                       | bitweises AND                                           |
| <code>^</code>                                                           | bitweises XOR                                           |
| <code> </code>                                                           | bitweises OR                                            |
| <code>&amp;&amp;</code>                                                  | logisches AND                                           |
| <code>  </code>                                                          | logisches OR                                            |
| <code>=, *=, /=, %=, +=, -=, &lt;&lt;=, &gt;&gt;=, &amp;=, ^=,  =</code> | Zuweisung                                               |

Außer die Zuweisungsoperatoren (in der letzten Tabellenzeile) werden alle Operatoren mit gleicher Priorität von links nach rechts abgearbeitet. Mit Klammerung ist es möglich, eine andere als die durch die Prioritäten vorgegebene Auswertung zu erzwingen.

Als Operanden sind Shellvariablen und Konstanten erlaubt. Für Shellvariablen muß nicht unbedingt das Integer-Attribut gesetzt sein, um sie in arithmetischen Ausdrücken verwenden zu können.

Konstanten haben die Form

`[basis#]zahl`

*basis* muß eine Zahl zwischen 2 und 36 sein und legt die Basis des Zahlensystems fest; Voreinstellung ist die Basis 10. *zahl* muß eine nicht-negative ganze Zahl sein. Bei einer Basis, die größer als 10 ist, sind Groß- oder Kleinbuchstaben zu verwenden, um eine Ziffer größer als 9 darzustellen. Beginnt eine *zahl* mit 0, wird sie als Oktalzahl interpretiert. Beginnt eine *zahl* mit **0x** oder **0X**, wird sie als Hexadezimalzahl interpretiert.

Um das Ergebnis eines arithmetischen Ausdrucks zu erhalten, ist entweder

`$(ausdr)` oder  
`$(((ausdr)))`

anzugeben. So gibt z.B. **echo \$(4+5\*6)** den Wert **34** aus. Der angegebene Ausdruck wird dabei so behandelt, als ob er in `"..."` angegeben wurden wäre, so daß innerhalb des Ausdrucks Parameter- und Kommandosubstitution möglich ist.

Schachtelung von arithmetischen Ausdrücken ist auch möglich. So gibt z.B. **echo \$((10+\$[4+5\*6]))** den Wert **44** aus.

Falls ein ungültiger Ausdruck angegeben wird, gibt die **bash** eine Fehlermeldung aus.

Das builtin-Kommando **let** erlaubt arithmetische Auswertungen für mehrere durch Leer- oder Tabzeichen getrennte Ausdrücke:

**let** *ausdruck* ...

Jeder angegebene *ausdruck* wird entsprechend den zuvor gegebenen Regeln ausgewertet. Der Rückgabewert von **let** ist 0 (erfolgreich), wenn die Auswertung des letzten Ausdrucks einen Wert verschieden von 0 liefert, andernfalls ist der Rückgabewert 1 (nicht erfolgreich).

## 7.17 Builtin-Kommandos

**:** [*argument(e)*]

Null-Kommando; identisch zu Bourne-Shell.

**.** *dateiname* [*argument(e)*]

Punkt-Kommando; identisch zu Bourne-Shell (siehe auch **source**-Kommando).

**alias** [*name*=[*wert*]...]

Ohne Argumente gibt **alias** alle momentan definierten Aliase aus.

**alias** *name*

gibt den Wert des Alias *name* aus, wenn ein solches existiert.

**alias** *name*=*wert*

definiert ein Alias *name* mit dem angegebenen *wert*. *wert* kann ein beliebiger Text sein. Für *wert* wird Parameter-, Kommandosubstitution und Dateinamenexpansion durchgeführt.

**bg** [*job(s)*]

**bg** bewirkt, daß die Ausführung der angehaltenen *job(s)* im Hintergrund fortgesetzt wird. Werden keine *job(s)* beim Aufruf angegeben, wird die Ausführung des aktuellen Jobs (zuletzt angehaltener Job) im Hintergrund fortgesetzt. Die einzelnen Jobs können über die PID, die Jobnummer oder den Jobnamen angesprochen werden. Um einen Job über seine Jobnummer oder seinen Namen anzusprechen, gibt es folgende Notationen:

**%jobnr**                      Job mit Jobnummer *jobnr*

**%string**                    Job, dessen Name mit *string* beginnt

**%?string**                  Job, dessen Name *string* enthält

**%+** oder **%%**      momentaner Job  
**%-**                    vorheriger aktueller Job

Die Angabe **%jobnr &** ist identisch zu **bg %jobnr**.

**bind** [-**m** *keymap*] [-**lvd**] [-**q** *name*]

**bind** [-**m** *keymap*] -**f** *dateiname*

**bind** [-**m** *keymap*] *tastenkombi: funktion-name*

zeigt die momentan vergebenen Tastenkombinationen an oder definiert neue für das Editieren von Kommandozeilen. Die Syntax für key-bindings ist dabei identisch zu der Syntax in der Datei *.inputrc*.

**-m** *keymap*            legt fest, ob key-bindings von emacs oder vi betroffen sind. Für *keymap* kann **emacs**, **emacs-standard**, **emacs-meta**, **emacs-ctlx**, **vi**, **vi-mode** oder **vi-insert** angegeben werden (**vi** ist äquivalent zu **vi-command**; **emacs** ist äquivalent zu **emacs-command**).

**-l**                      listet alle Namen der möglichen key-bindings auf.

**-v**                      listet die key-bindings mit den zugeordneten Tastenkombinationen auf.

**-d**                      listet die key-bindings so auf, daß sie in einer *.inputrc* abgelegt werden könnten.

**-f** *dateiname*            liest die key-bindings aus Datei *dateiname*.

**-q** *name*                gibt die key-bindings aus, die der Funktion *name* zugeordnet sind.

**break** [*n*]

Verlassen einer **for**-, **while**-, **until**- bzw. **select**-Schleife; identisch zu Korn-Shell.

**builtin** [*builtin-kdo* [*argument(e)*]]

**builtin** startet das *builtin-kdo*. **builtin** wird benötigt, wenn ein builtin-Kommando durch eine Funktion ersetzt wird, aber innerhalb der Funktion das entsprechende *builtin-kdo* aufzurufen ist.

**cd** [*directory*]

Wechseln in ein anderes Working-Directory; identisch zu Korn-Shell. Angabe von - für *directory* ist identisch zu **\$OLDPWD**.

**command** [-pVv] *kdo* [*argument(e)*]

**command** startet das Kommando *kdo* mit den angegebenen *argumente*, selbst wenn es eine gleichnamige Funktion mit Namen *kdo* gibt. Wenn man z.B. eine eigene Funktion **rm** definiert hat, und man möchte das Kommando **rm** und nicht die Funktion **rm** starten, muß man **command rm** aufrufen.

Ist die Option **-p** angegeben, wird für das Suchen von *kdo* die Default-Einstellung von **PATH** verwendet. So ist garantiert, daß alle Standard-Kommandos gefunden werden.

Ist eine der Optionen **-v** oder **-V** angegeben, wird eine kurze Beschreibung von *kdo* ausgegeben. Bei **-V** ist diese Beschreibung etwas ausführlicher als bei **-v**, wie z.B.

```
$ command -V ls
ls is aliased to `ls -F`
$ command -v ls
alias ls=`ls -F`
$ command -V rm
rm is /bin/rm
$ command -v rm
/bin/rm
$
```

**continue** [*n*]

Abbrechen eines **for**-, **while**-, **until**- bzw. **select**-Schleifendurchlaufs; identisch zu Korn-Shell.

**declare** [-frxi] [*name*=[*wert*]] ...

**declare** deklariert Variablen und/oder teilt ihnen Attribute zu. Sind keine *namen* angegeben, werden die Werte aller Variablen ausgegeben.

Ist Option **-f** angegeben, bezieht sich **declare** nur auf Funktionsnamen.

Option **-r** legt fest, daß die entsprechenden *namen* nur lesbar sein sollen.

Option **-x** legt fest, daß die entsprechenden *namen* für den Export zu markieren sind.

Option **-i** legt fest, daß die entsprechenden *namen* als Integer-Variablen zu behandeln sind. Bei jeder Zuweisung eines Wertes an diese Variablen wird dann eine arithmetische Auswertung durchgeführt.

Wird bei den Optionen anstelle eines Minus (-) ein Plus (+) angegeben, wird das entsprechende Attribut nicht ein-, sondern ausgeschaltet.

Wird **declare** innerhalb einer Funktion verwendet, werden hiermit lokale *namen* deklariert, was auch mit dem Kommando **local** möglich ist.

**dirs** [-l] [+/-n]

Ausgeben des Directorystacks; ähnlich zur *csh*. Ist *+n* angegeben, wird nur der *n*.te Eintrag (von links) im Directorystack angezeigt. Ist *-n* angegeben, wird nur der *n*.te Eintrag (von rechts) im Directorystack angezeigt. Ist Option *-l* angegeben, wird bei der Ausgabe Tilde-Expandierung durchgeführt.

**echo** [-neE] [*argument(e)*]

Ausgeben der Argumente als Text (durch Leerzeichen getrennt); weitgehend identisch zur C-Shell. Ist *-n* angegeben, wird die Ausgabe nicht mit einem Neuzeile-Zeichen abgeschlossen. Ist *-e* angegeben, haben die von C her bekannten Escape-Sequenzen (*\t*, *\n*, *\a*, ..) ihre Sonderbedeutung. Ist *-E* angegeben, verlieren diese C-Escape-Sequenzen ihre Sonderbedeutung.

**enable** [-n] [-a] [*name(n)*]

**enable** ermöglicht es, builtin-Kommandos ein- oder auszuschalten. So ist es möglich, anstelle eines builtin-Kommandos ein gleichnamiges anderes Programm aufzurufen, wenn das betreffende builtin-Kommando ausgeschaltet ist.

Ist Option *-n* angegeben, werden die Kommandos *name(n)* ausgeschaltet. Um z.B. ein Programm mit dem Namen **test** (anstelle des builtin-Kommandos) aufrufen zu können, müßte zuvor **enable -n test** eingegeben werden. Mit **enable** *name(n)* können entsprechende builtin-Kommandos wieder eingeschaltet werden.

Ist Option *-a* angegeben, werden alle builtin-Kommandos aufgelistet, wobei zu jedem einzelnen builtin-Kommando angegeben ist, ob es zur Zeit eingeschaltet (*enable*) oder ausgeschaltet ist (*enable -n*).

Sollen nur die momentan eingeschalteten builtin-Kommandos angezeigt werden, muß **enable** ohne Optionen und *name(n)* aufgerufen werden.

**eval** [*argument(e)*]

Ausführen der Argumente als Kommandos; identisch zu Bourne-Shell.

**exec** [[-] *kdo* [*argument(e)*]]

Überlagern der Shell mit einem Kommando bzw. Ein-/Ausgaben der aktuellen Shell umlenken; weitgehend identisch zu Korn-Shell.

**exit** [*n*]

Beenden der momentan aktiven Shell; identisch zur Bourne-Shell.

**export** [-nf] [name[=wert]] ....

**export -p**

Exportieren von Shellvariablen; weitgehend identisch zur Korn-Shell. Ist **-f** angegeben, handelt es sich bei den *namen* um Funktionsnamen. Sind keine *namen* oder ist die Option **-p** angegeben, wird eine Liste aller zum Export markierten Namen ausgegeben. Mit der Option **-n** wird das Exportieren für die angegebenen *namen* wieder aufgehoben.

**fc -e -s** [alt=neu] [kommando]

**fc** [-e editor] [-nlr] [von [bis]]

Erneutes Ausführen bzw. Auflisten zuvor eingegebener Kommandozeilen; weitgehend identisch zur Korn-Shell.

**fg** [job(s)]

**fg** bewirkt, daß die Ausführung der angehaltenen *job(s)* bzw. der Hintergrund-*job(s)* der Reihe nach im Vordergrund fortgesetzt wird. Werden keine *job(s)* beim Aufruf angegeben, wird die Ausführung des aktuellen Jobs (zuletzt angehaltener Job oder zuletzt im Hintergrund gestarteter Job) im Vordergrund fortgesetzt. Der aktuelle Job kann auch mit dem Kommando **jobs** ermittelt werden. Für *job(s)* sind die folgenden Angaben möglich:

|                   |                                            |
|-------------------|--------------------------------------------|
| <b>%jobnr</b>     | Job mit Jobnummer <i>jobnr</i>             |
| <b>%string</b>    | Job, dessen Name mit <i>string</i> beginnt |
| <b>%?string</b>   | Job, dessen Name <i>string</i> enthält     |
| <b>%+ oder %%</b> | momentaner Job                             |
| <b>%-</b>         | vorheriger aktueller Job                   |

Die Angabe **%jobnr** ist identisch zu **fg %jobnr**.

**getopts** opstring name [argument(e)]

Auswerten der Optionen in einem Shell-Skript oder Funktion; weitgehend identisch zur Bourne-Shell. Neu ist, daß mit dem Setzen von **OPTERR** auf 0 das Ausgeben von Fehlermeldungen bei falscher Optionenangabe unterdrückt werden kann.

**hash** [-r] [name(n)]

Abfragen bzw. Ändern der internen hashing-Tabelle; identisch zur Bourne-Shell.

**help** [pattern]

**help** gibt zu allen builtin-Kommandos, die durch *pattern* abgedeckt werden, eine Beschreibung aus. Wird **help** ohne *pattern* aufgerufen, so listet es alle builtin-Kommandos mit einer Kurzbeschreibung auf.



**history** [*n*]**history -rwan** [*historydatei*]

Ausgeben von Kommandos aus der History-Liste. Ohne Optionen werden alle Kommandos aus der History-Liste mit Zeilennummern ausgegeben. Mit \* markierte Zeilen wurden modifiziert. Ist eine Zahl *n* angegeben, werden nur die letzten *n* Zeilen aus der History-Liste ausgegeben.

Ist ein *historydatei* angegeben, bezieht sich **history** auf diese History-Datei, andernfalls bezieht sich **history** auf die History-Datei, die über die Variable **HISTFILE** festgelegt ist.

- a Neue (seit **bash**-Start eingegebenen) Kommandozeilen an History-Liste anhängen.
- n Noch nicht aus History-Datei gelesene Zeilen in aktuelle History-Liste aufnehmen. Dies sind Kommandozeilen, die seit dem Start der **bash** lediglich an die History-Datei angehängt wurden.
- r Inhalt der History-Datei als neue History-Liste verwenden.
- w Überschreiben der History-Datei mit der aktuellen History-Liste.

**jobs** [-l*pn*] [*job(s)*]**jobs -x** *kommando* [*job(s)*]

Mit dem ersten **jobs**-Aufruf können Informationen zu den angegebenen *job(s)* oder allen momentan vorhandenen Jobs (wenn *job(s)* nicht angegeben ist) abgefragt werden.

- l nach Jobnummer wird noch PID des Jobs ausgegeben.
- p nur die PIDs der Prozeßgruppenführer ausgegeben.
- n nur Jobs ausgegeben, deren Status sich seit letztem Aufruf geändert hat.

Beim zweiten **jobs**-Aufruf (-x angegeben) wird *kommando* ausgeführt, wobei ihm als Argumente die PIDs der *job(s)* übergeben werden. So würde z.B. der folgende Aufruf ein zuvor gestartetes und im Hintergrund ablaufendes **find**-Kommando abbrechen:

**jobs -x kill -9 %find**

Für *job(s)* sind die folgenden Angaben möglich:

- |                  |                                            |
|------------------|--------------------------------------------|
| % <i>jobnr</i>   | Job mit Jobnummer <i>jobnr</i>             |
| % <i>string</i>  | Job, dessen Name mit <i>string</i> beginnt |
| %? <i>string</i> | Job, dessen Name <i>string</i> enthält     |
| %+ oder %%       | momentaner Job                             |
| %-               | vorheriger aktueller Job                   |

**kill** [-s *signal* | -*signal*] [*pid* | *job*] ....

**kill -l** [*signr*]

Aufrufe sind weitgehend identisch zur Korn-Shell. Die erste Aufrufform ermöglicht das Schicken eines Signals *signal* an einen mit *pid* oder *job* ausgewählten Prozeß. Für *signal* kann entweder ein Signalname (wie **SIGHUP**, **SIGKILL**, ..) oder eine Signalnummer angegeben werden. Ein Signalname kann dabei in Groß- oder Kleinschreibung angegeben werden und das Präfix **SIG** kann auch weggelassen werden. Ist kein *signal* angegeben, wird **SIGTERM** angenommen.

Bei der zweiten Aufrufform (**-l**) werden entweder alle Signalnamen mit Signalnummern oder nur der Signalname mit der Signalnummer *signr* angegeben.

**let** *argument(e)*

Auswerten von arithmetischen Ausdrücken *argument(e)*; identisch zur Korn-Shell.

**local** *name*[=*wert*] ...

**local**, das nur innerhalb von Funktionsdefinitionen verwendet werden darf, deklariert funktionslokale Variablen. Sind keine Argumente angegeben, listet **local** alle funktionslokalen Variablen.

**logout**

Login-Shell beenden; ähnlich zur *csh*.

**popd** [+/-*n*]

Ähnlich zur *csh*. Wird **popd** ohne Argument aufgerufen, wird das an zweit oberster Stelle des Directorystacks stehende Directory zum neuen Working-Directory; der an oberster Stelle des Directorystacks stehende Pfad wird dabei entfernt. Wird *+n* angegeben, wird der *n*.te Pfad aus dem Directorystack entfernt (*n*=0 entfernt den ersten Eintrag, *n*=1 den zweiten, usw.). Wird *-n* angegeben, wird der *n*.te Pfad (von rechts) aus dem Directorystack entfernt (**popd -0** entfernt den letzten Eintrag, **popd -1** den vorletzten, usw.).

**pushd** [*dir*]

**pushd** +/-*n*

Ähnlich zur *csh*. Hinzufügen eines Directorys zum Directorystack bzw. Vertauschen oder Rotieren der Directories im Directorystack, wobei das oberste Directory des Directorystacks immer auch automatisch das Working-Directory wird. Wird **pushd** ohne Argumente aufgerufen, werden die beiden obersten Einträge des Directorystacks vertauscht. Wird *+n* angegeben, wird der Directorystack so rotiert, daß der *n*.te Pfad das oberste Element des Directorystacks wird. Wird *-n* angegeben, wird der Directorystack so rotiert,

daß der *n*.te Pfad (von rechts) das oberste Element des Directorystacks wird. Ist *dir* angegeben, wird dieses Directory das neue Working-Directory und an oberster Stelle des Directorystacks abgelegt.

### **pwd**

Ausgeben des aktuellen Working-Directorys; identisch zu Bourne-Shell.

### **read -r** [*name(n)*]

Lesen einer Eingabezeile von der Standardeingabe. Das erste Wort wird dabei dem ersten *namen*, das zweite Wort dem zweiten *namen*, usw. zugewiesen. Sind mehr Worte als *namen* vorhanden, wird der Rest der Zeile dem letzten *namen* zugewiesen. Worttrennzeichen werden dabei über **IFS** festgelegt. Sind keine *namen* angegeben, wird die gelesene Zeile in der Variablen **REPLY** abgelegt. Die Option **-r** schaltet die Sonderbedeutung von \ als Zeilen-Fortsetzungszeichen aus.

### **readonly [-f]** [*name(n)*]

#### **readonly -p**

Shellvariablen oder Shell-Funktionen (**-f**) als »nur lesbar« kennzeichnen. Sind keine Argumente und die Option **-p** angegeben, werden alle als *readonly* markierten Namen ausgegeben.

### **return** [*n*]

Verlassen einer Funktion mit Rückgabewert *n*; identisch zur Bourne-Shell. Ist *n* nicht angegeben, ist der Rückgabewert der Funktion der exit-Status des letzten Kommandos. Wird beim Lesen einer Datei mit . (Punkt) oder **source** ein **return** gelesen, beendet dieses das Lesen.

### **set [-abefhkmnptuvxldCHP-] [-o optname]... [argument(e)]**

Setzen von **bash**-Optionen oder Positionsparametern.

- a** (**allexport**) Shellvariablen, die verändert oder neu angelegt werden, für den Export markieren.
- b** (**notify**) Status von beendeten Hintergrund-Jobs sofort und nicht erst bei Ausgabe des nächsten Prompts melden.
- e** (**errexit**) Shell verlassen, wenn ein einfaches Kommando nicht erfolgreich ausgeführt werden kann.
- f** (**noglob**) Dateinamenexpansion ausschalten.
- h** (**trackall**) Kommandos bereits bei Definition einer Funktion in Hash-tabelle eintragen und nicht erst, wenn Funktion aufgerufen wird.

- k** (**keyword**) Alle Shellvariablen an die Umgebung (*Environment*) eines Kommandos übergeben. Normalerweise werden Variablen-Zuweisungen nur dann an aufgerufene Kommandos übergeben, wenn sie beim Aufruf vor dem Kommandonamen angegeben sind.
- m** (**monitor**) Jobkontrolle einschalten.
- n** (**noexec**) Kommandos nur lesen und auf Syntaxfehler untersuchen, aber nicht ausführen. **-n** wird bei nicht interaktiven Shells ignoriert.
- o optname** Für *optname* kann dabei folgendes angegeben werden:
  - allexport** entspricht **-a**
  - braceexpand** Dateinamenexpansion {*wort*<sub>1</sub>,*wort*<sub>2</sub>,..} einschalten
  - emacs** **emacs** für Kommandozeilen-Editierung einschalten
  - errexit** entspricht **-e**
  - histexpand** entspricht **-H**
  - ignoreeof** Eine interaktive Shell kann nur mit **exit** (nicht mit *Strg-d*) verlassen werden; entspricht **IGNOREEOF=10**.
  - interactive-comments** Kommentar (#) in interaktiver Shell zulassen.
  - monitor** entspricht **-m**
  - noclobber** entspricht **-C**
  - noexec** entspricht **-n**
  - noglob** entspricht **-f**
  - nohash** entspricht **-d**
  - notify** entspricht **-b**
  - nounset** entspricht **-u**
  - physical** entspricht **-P**
  - posix** bash verhält sich entsprechend POSIX 1003.2.
  - privileged** entspricht **-p**
  - verbose** entspricht **-v**
  - vi** **vi** für Kommandozeilen-Editierung einschalten
  - xtrace** entspricht **-x**

- p** Privilegierten Modus einschalten. In diesem Modus wird die Variable **ENV** nicht ausgewertet und Funktionen werden nicht aus der Umgebung vererbt. **-p** wird automatisch beim Startup eingeschaltet, wenn die effektive user(group)-ID verschieden von der realen user(group)-ID ist. Wird diese Option ausgeschaltet, werden die effektive user- und group-IDs auf die entsprechenden realen IDs gesetzt.
- t** Nach Ausführung eines Kommandos **bash** beenden.
- u** (**nounset**) Zugriff auf nicht gesetzte Shellvariable als Fehler werten.
- v** (**verbose**) Alle Shell-Eingabezeilen vor ihrer Ausführung – so wie sie gelesen werden – ausgeben.
- x** (**xtrace**) Jedes einfache Kommando unmittelbar vor seiner Ausführung (nach Substitutionen und Expandierungen) mit vorangestelltem **\$PS4** anzeigen.
- d** Aufgerufene Kommandos nicht in Hashingtabelle ablegen.
- C** Ausgabeumlenkung in existierende Dateien unterbinden.
- H** History-Mechanismus (Ausrufezeichen !) einschalten (Voreinstellung: **-H** bei interaktiver **bash** eingeschaltet).
- P** Symbolischen Links nicht folgen.
- (Minuszeichen) zunächst die beiden Optionen **-x** und **-v** ausschalten und Rest der Kommandozeile als normale Argumente interpretieren, selbst wenn sie mit **-** (Minus) beginnen.
- (doppeltes Minuszeichen) hat keine Auswirkung auf die Optionen. Wenn nach **--** keine weiteren Argumente angegeben sind, werden die Positionsparameter gelöscht, ansonsten werden diese als Positionsparameter genommen, selbst wenn sie mit **-** beginnen.

Wird den Optionen anstelle eines Minus (-) ein Plus (+) vorangestellt, wird entsprechende Option ausgeschaltet.

**\$-** enthält alle momentan gesetzten **bash**-Optionen.

Angegebene *argument(e)* sind Positionsparameter. Sind keine *argument(e)* angegeben, werden alle momentan definierten Shellvariablen ausgegeben.

### **shift** [*n*]

Verschieben der Werte von Positionsparametern; identisch zur Bourne-Shell.

### **source** *dateiname* [*argument(e)*]

verhält sich wie das Punkt-Kommando: In der aktuellen **bash** werden die Kommandos aus der Datei *dateiname* gelesen und ausgeführt; ähnlich zur **csh**.

## **suspend [-f]**

hält die aktuelle Shell an, bis sie das Signal **SIGCONT** empfängt. Ist Option **-f** angegeben, wird keine Warnung ausgegeben, wenn es sich bei aktueller Shell um eine Login-Shell handelt.

## **test *ausdr***

**[ *ausdr* ]**

Auswerten eines Ausdrucks; weitgehend identisch zu Bourne-Shell.

**test** wertet den angegebenen Ausdruck *ausdr* aus. Ist die über *ausdr* angegebene Bedingung erfüllt, so liefert **test** bzw. **[.]** den exit-Status 0 (wahr oder erfolgreich), ansonsten einen von 0 verschiedenen Wert (falsch oder nicht erfolgreich). Sind keine Argumente angegeben, dann liefert **test** einen exit-Status verschieden von 0 (falsch). Es können die folgenden Ausdrücke für *ausdr* angegeben werden:

| Ausdruck                   | liefert wahr (exit-Status 0), wenn                                                                                                                                                          |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-r</b> <i>datei</i>     | <i>datei</i> existiert und gelesen ( <b>r</b> ead) werden darf.                                                                                                                             |
| <b>-w</b> <i>datei</i>     | <i>datei</i> existiert und beschrieben ( <b>w</b> rite) werden darf.                                                                                                                        |
| <b>-x</b> <i>datei</i>     | <i>datei</i> existiert und ausführbar ( <b>e</b> xecute) ist.                                                                                                                               |
| <b>-f</b> <i>datei</i>     | <i>datei</i> existiert und eine normale Datei ( <b>f</b> ile) ist.                                                                                                                          |
| <b>-d</b> <i>datei</i>     | <i>datei</i> existiert und ein Directory ( <b>d</b> irectory) ist.                                                                                                                          |
| <b>-c</b> <i>datei</i>     | <i>datei</i> existiert und eine "zeichenspezifische Gerätedatei" ( <b>c</b> harter) ist.                                                                                                    |
| <b>-b</b> <i>datei</i>     | <i>datei</i> existiert und eine "blockspezifische Gerätedatei" ( <b>b</b> lock) ist.                                                                                                        |
| <b>-p</b> <i>datei</i>     | <i>datei</i> existiert und eine named <b>p</b> ipe ist.                                                                                                                                     |
| <b>-u</b> <i>datei</i>     | <i>datei</i> existiert und das set- <b>u</b> ser-id Bit gesetzt ist.                                                                                                                        |
| <b>-g</b> <i>datei</i>     | <i>datei</i> existiert und das set- <b>g</b> roup-id Bit gesetzt ist.                                                                                                                       |
| <b>-k</b> <i>datei</i>     | <i>datei</i> existiert und das sticky-Bit gesetzt ist.                                                                                                                                      |
| <b>-s</b> <i>datei</i>     | <i>datei</i> existiert und nicht leer ( <b>s</b> pace) ist.                                                                                                                                 |
| <b>-t</b> [ <i>fd</i> ]    | die geöffnete Datei, deren Filedeskriptor <i>fd</i> ist, der Dialogstation ( <b>t</b> erminal) zugeordnet ist. Ist <i>fd</i> nicht angegeben, wird hierfür der Filedeskriptor 1 angenommen. |
| <b>-z</b> <i>zkt</i>       | die Länge der Zeichenkette <i>zkt</i> gleich 0 ( <b>z</b> ero) ist.                                                                                                                         |
| <b>-n</b> <i>zkt</i>       | die Länge der Zeichenkette <i>zkt</i> nicht gleich 0 ( <b>n</b> ot zero) ist.                                                                                                               |
| <i>zkt1</i> = <i>zkt2</i>  | die Zeichenketten <i>zkt1</i> und <i>zkt2</i> identisch sind.                                                                                                                               |
| <i>zkt1</i> != <i>zkt2</i> | die Zeichenketten <i>zkt1</i> und <i>zkt2</i> verschieden sind.                                                                                                                             |
| <i>zkt</i>                 | <i>zkt</i> nicht die leere Zeichenkette ist.                                                                                                                                                |

| Ausdruck                       | liefert wahr (exit-Status 0), wenn                                                                  |
|--------------------------------|-----------------------------------------------------------------------------------------------------|
| <i>n1</i> <b>-eq</b> <i>n2</i> | die ganzen Zahlen <i>n1</i> und <i>n2</i> gleich ( <b>equal</b> ) sind.                             |
| <i>n1</i> <b>-ne</b> <i>n2</i> | die ganzen Zahlen <i>n1</i> und <i>n2</i> nicht gleich ( <b>not equal</b> ) sind.                   |
| <i>n1</i> <b>-gt</b> <i>n2</i> | die ganze Zahl <i>n1</i> größer als ( <b>greater than</b> ) die ganze Zahl <i>n2</i> ist.           |
| <i>n1</i> <b>-ge</b> <i>n2</i> | die ganze Zahl <i>n1</i> größer oder gleich ( <b>greater equal</b> ) der ganzen Zahl <i>n2</i> ist. |
| <i>n1</i> <b>-lt</b> <i>n2</i> | die ganze Zahl <i>n1</i> kleiner als ( <b>less than</b> ) die ganze Zahl <i>n2</i> ist.             |
| <i>n1</i> <b>-le</b> <i>n2</i> | die ganze Zahl <i>n1</i> kleiner oder gleich ( <b>less equal</b> ) der ganzen Zahl <i>n2</i> ist.   |

Alle diese Ausdrücke können mit den folgenden **Operatoren** zu neuen Ausdrücken verknüpft werden:

| Ausdruck                              | Bedeutung                   |
|---------------------------------------|-----------------------------|
| <b>!</b> <i>ausdr</i>                 | Negationsoperator.          |
| <i>ausdr1</i> <b>-a</b> <i>ausdr2</i> | AND-Operator.               |
| <i>ausdr1</i> <b>-o</b> <i>ausdr2</i> | OR-Operator.                |
| ( <i>ausdr</i> )                      | Klammerung eines Ausdrucks. |

Zusätzlich können in der **bash** noch folgende Ausdrücke angegeben werden:

| Ausdruck                           | liefert wahr (exit-Status 0), wenn                                                                      |
|------------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>-f</b> <i>datei</i>             | <i>datei</i> existiert ( <b>exist</b> ).                                                                |
| <b>-L</b> <i>datei</i>             | <i>datei</i> existiert und ein symbolischer Link ( <b>Link</b> ) ist.                                   |
| <b>-O</b> <i>datei</i>             | <i>datei</i> existiert und der <i>datei</i> -Eigentümer der effektiven UID entspricht ( <b>O</b> wner). |
| <b>-G</b> <i>datei</i>             | <i>datei</i> existiert und die <i>datei</i> -Gruppe der effektiven GID entspricht ( <b>G</b> roup).     |
| <b>-S</b> <i>datei</i>             | <i>datei</i> existiert und eine spezielle Datei vom <i>socket</i> -Typ ist ( <b>Sok</b> ket).           |
| <i>dat1</i> <b>-nt</b> <i>dat2</i> | die Datei <i>dat1</i> neuer als Datei <i>dat2</i> ist ( <b>newer than</b> ).                            |
| <i>dat1</i> <b>-ot</b> <i>dat2</i> | die Datei <i>dat1</i> älter als Datei <i>dat2</i> ist ( <b>older than</b> ).                            |
| <i>dat1</i> <b>-ef</b> <i>dat2</i> | die Datei <i>dat1</i> nur ein anderer Name für die Datei <i>dat2</i> ist ( <b>equal file</b> ).         |

Wenn in den obigen Ausdrücken für *datei* der Pfadname */dev/fd/n* angegeben ist, wird der Filedeskriptor *n* geprüft.

**times**

Anzeigen der von der Shell verbrauchten CPU-Zeit; identisch zu Bourne-Shell.

**trap [-l] [argument] [signale(e)]**

Installieren bzw. Auflisten von Signalhandlern; weitgehend identisch zur Korn-Shell.

**trap** legt die Reaktion der Shell für asynchron eintreffende Signale fest. Trifft ein Signal ein, das in der *signale(e)*-Liste angegeben ist, dann führt die Shell die als *argument* angegebenen Kommandos aus und danach setzt sie ihre Ausführung an der Stelle fort, an der die durch das Signal bedingte Unterbrechung stattfand. Als Signalbehandlung ist dabei möglich:

- 1) Es kann für *argument* eine Liste von Kommandos angegeben werden ('*kdoliste*'), die bei Eintreffen eines der über *signale(e)* spezifizierten Signale auszuführen ist.
- 2) Ignorieren der Signale *signale(e)*, indem für *argument* eine leere Zeichenkette (z.B. "" oder ") angegeben wird.
- 3) Wird kein *argument* (oder -) angegeben, wird für die Signale *signale(e)* wieder die vom System voreingestellte Signalbehandlung festgelegt.

Werden beim Aufruf von **trap** kein *argument* und keine *signale(e)* angegeben, dann gibt **trap** die Signalnummern aus, für die momentan mit Hilfe eines **trap**-Kommandos eine benutzerspezifische Signalbehandlung eingestellt wurde.

Wird **trap -l** aufgerufen, wird eine Liste alle Signalnamen mit ihren zugehörigen Nummern ausgegeben.

Neu in der **bash** ist, daß die Signale nicht nur über ihre Nummern (aus *<signal.h>* oder *<sys/signal.h>*), sondern auch über symbolische Namen angesprochen werden können. Die symbolischen Namen der am häufigsten verwendeten Signale sind:

|             |                   |
|-------------|-------------------|
| <b>HUP</b>  | <i>hangup.</i>    |
| <b>INT</b>  | <i>intr.</i>      |
| <b>QUIT</b> | <i>quit.</i>      |
| <b>TERM</b> | <i>terminate.</i> |
| <b>KILL</b> | <i>kill.</i>      |



|              |                                                                                                                              |
|--------------|------------------------------------------------------------------------------------------------------------------------------|
| <b>EXIT</b>  | <i>exit</i> : wird beim Verlassen einer Funktion oder einer <b>bash</b> erzeugt.                                             |
| <b>ERR</b>   | <i>error</i> : wird immer dann geschickt, wenn ein Kommando einen exit-Status verschieden von 0 (nicht erfolgreich) liefert. |
| <b>DEBUG</b> | <i>debugging</i> : wird nach jeder Ausführung eines Kommandos erzeugt.                                                       |

### **trap - SIGINT SIGQUIT SIGEXIT**

Die Signalbehandlung für die Signale **INT**, **QUIT** und **EXIT** wird auf die vom System voreingestellte Signalbehandlung zurückgesetzt.

### **trap INT QUIT EXIT**

gleiche Auswirkung wie der vorherige Aufruf.

**type [-all] [-type|-path] name...**

**type** gibt aus, welche Programme ausgeführt werden, wenn die angegebenen Kommando-*name(n)* aufgerufen würden. Sind keine Optionen angegeben, gibt **type** für jeden einzelnen *namen* aus, ob es sich dabei um ein Alias, Schlüsselwort, Funktion, builtin-Kommando oder um eine Programmdatei handelt, wie z.B.

```
$ type more
more is /bin/more
$ type m
m is aliased to 'less -E '
$ type for test
for is a shell keyword
test is a shell builtin
```

Ist Option **-type** angegeben, gibt **type** nur **alias**, **keyword**, **function**, **builtin** oder **file** für die angegebenen *namen* aus.

```
$ type -type more
file
$ type -type m
alias
$ type -type for test
keyword
builtin
```

Ist Option **-path** angegeben, gibt **type** nur dann etwas aus, nämlich den Pfadnamen, wenn es sich bei einem *namen* um eine Programmdatei handelt.

```
$ type -path more
/bin/more
$ type -path m
$ type -path for test
```

Ist Option **-all** angegeben, gibt **type** alle Vorkommen (Alias, Schlüsselwort, Funktion, builtin-Kommando, Programmdatei) von den angegebenen *namen* aus.

```
$ type -all more
more is /bin/more
more is /usr/bin/more
$ type -all ls
ls is aliased to '/bin/ls $LS_OPTIONS '
ls is /bin/ls
```

**typeset [-frxi] [name=[wert]] ...**

Ein-/Ausschalten bzw. Anzeigen von Attributen für Variablen oder für Funktionen; völlig identisch zum builtin-Kommando **declare** (siehe vorher).

**ulimit [-acdmstfpnuvSH] [limit]**

Mit **ulimit** ist es möglich, Limits für Systemressourcen festzulegen oder sich die momentan gesetzten Limits dafür anzeigen zu lassen. Die Limits gelten dabei für die aktuelle Shell und für jeden ihrer Kindprozesse.

- S** (*Softlimit*) Ändern / Anzeigen von Softlimits (Voreinstellung, wenn **-H** nicht angegeben). Softlimits können nur bis zum Wert des entsprechenden Hardlimits erhöht werden.
- H** (*Hardlimit*) Ändern / Anzeigen von Hardlimits; Hardlimits können nicht wieder erhöht werden.
- a** (*all*) Anzeigen aller momentan gültigen Limits.
- c** (*core*) Maximale Größe für *core*-Dateien.
- d** (*data segment*) Maximale Größe des Datensegments.
- f** (*file*) Maximale Größe von zu schreibenden Dateien.
- m** (*memory*) Maximale Größe für physikalischen Hauptspeicher.
- n** (*number*) Maximale Anzahl von offenen Filedeskriptoren.
- p** (*pipe*) Maximale Größe für den Puffer einer Pipe (Anzahl der 512-Byte-Blöcke).
- s** (*stack segment*) Maximale Größe des Stacksegments.
- t** (*time*) Limit für die CPU-Zeit in Sekunden.
- u** (*user process*) Maximale Anzahl von Prozessen für einen Benutzer.
- v** (*virtual memory*) Maximale Größe von virtuellem Speicher.

Ist *limit* nicht angegeben, wird das momentan festgelegte Limit ausgegeben. Jeder Benutzer kann sein ihm zugestandenes Limit erniedrigen. Erhöhen eines Hardlimits ist jedoch nur dem Superuser vorbehalten.

Ist *limit* angegeben, wird damit ein Limit für die entsprechende Systemresource festgelegt. Für *limit* kann ein beliebiger arithmetischer Ausdruck oder **unlimited** (unbegrenzt) angegeben werden. Die für *limit* angegebenen Werte werden mit 1024 multipliziert, außer für **-t** (Sekundenzahl), **-p** (multipliziert mit 512), für **-n** und **-u**, wo sie die wirklichen Werte darstellen.

Ist keine Option angegeben, so entspricht dies der Angabe von **-f**.

### **umask** [-S] [*modus*]

Setzen bzw. Anzeigen der Dateikreierungsmaske. Beginnt *modus* mit einer Ziffer, wird er als eine Oktalzahl interpretiert. Ansonsten wird er als ein symbolischer Modus (ähnlich zu **chmod**) für die zu setzende Dateikreierungsmaske interpretiert.

Ist kein *modus* angegeben, wird die aktuelle Dateikreierungsmaske ausgegeben. Ist in diesem Fall die Option **-S** angegeben, wird die Dateikreierungsmaske in symbolischer Form ausgegeben.

```
$ umask
022
$ umask -S
u=rwx,g=rX,o=rX
```

### **unalias** [-a] *name(n)*

**unalias** löscht die angegebenen Aliase *name(n)*. Ist die Option **-a** angegeben, werden alle momentan definierten Aliase gelöscht.

### **unset** [-fv] [*name(n)*]

Löschen von Shellvariablen (**-v**) oder Shell-Funktionen (**-f**). Ist keine der Optionen angegeben, ist gleichzeitiges Löschen von Variablen und Funktionen möglich. **unset** kann nicht auf die vordefinierten Shellvariablen **PATH**, **IFS**, **PPID**, **PS1**, **PS2**, **UID** und **EUID** angewendet werden. Wird eine der Variablen **RANDOM**, **SECONDS**, **LINENO** oder **HISTCMD** gelöscht, verlieren sie ihre Sonderbedeutung, und diese kann auch nicht wieder eingeschaltet werden.

### **wait** [*job(s)*]

Warten auf die Beendigung von Subshells; identisch zur Korn-Shell. **wait** veranlaßt die **bash**, auf die Beendigung der angegebenen *job(s)* zu warten. Sind keine *job(s)* angegeben, wird auf die Beendigung aller Kindprozesse der momentan aktiven **bash** gewartet. Die einzelnen Jobs können über die PID, die Jobnummer oder den Jobnamen angesprochen werden. Um einen Job über seine Jobnummer oder seinen Namen anzusprechen, gibt es folgende Notationen:

|                         |                                            |
|-------------------------|--------------------------------------------|
| <code>%jobnr</code>     | Job mit Jobnummer <i>jobnr</i>             |
| <code>%string</code>    | Job, dessen Name mit <i>string</i> beginnt |
| <code>??string</code>   | Job, dessen Name <i>string</i> enthält     |
| <code>%+ oder %%</code> | aktueller Job                              |
| <code>%-</code>         | vorheriger aktueller Job                   |

Für *job(s)* können auch die PIDs von Prozessen angegeben werden, um auf deren Beendigung zu warten. Der exit-Status von **wait** ist der exit-Status des letzten Prozesses, auf den gewartet wurde.

## 7.18 Startup-Dateien der bash

Hier zeigt die **bash** unterschiedliches Verhalten für Login-Shells, interaktive Shells, die keine Login-Shell sind, und nicht interaktive Shells.

Eine *Login-Shell* ist eine Shell, bei der **\$0** (Name der Shell) mit einem Minuszeichen (wie **-bash**) beginnt, oder die Option **-login** beim Aufruf gesetzt wurde. Eine *interaktive Shell* ist eine Shell, bei der sowohl die Standardeingabe als auch die Standardausgabe auf ein Terminal eingestellt ist, oder eine Shell, die mit Option **-i** gestartet wurde.

### Login-Shell

Beim **Anmelden** gilt, wenn **-noprofile** nicht gesetzt ist:

```
if /etc/profile existiert
then Lies Inhalt und führe ihn aus

if ~/.bash_profile existiert
then Lies Inhalt und führe ihn aus
else if ~/.bash_login existiert
 then Lies Inhalt und führe ihn aus
 else if ~/.profile existiert
 then Lies Inhalt
```

Die Datei `./bash_profile` enthält meist die folgende Zeile:

```
if [-f ~/.bashrc]; then source ~/.bashrc; fi
```

Wenn die **bash** mit **sh** aufgerufen wird, verhält sie sich weitgehend wie die Bourne-Shell und liest nur den Inhalt der Startup-Dateien `/etc/profile` und `~/.profile`, außer es ist **-noprofile** gesetzt.

Beim Abmelden gilt:

```
if ~/.bash_logout existiert
then Lies Inhalt und führe ihn aus
```

### Interaktive Shells, die keine Login-Shells sind

Beim Anmelden gilt, wenn **-norc** und **-rcfile** nicht gesetzt sind:

```
if ~/.bashrc existiert
then Lies Inhalt und führe ihn aus
```

Um in einer Startup-Datei festzustellen, ob eine Shell interaktiv ist oder nicht, muß die Variable **PS1** geprüft werden. Diese Variable **PS1** wird nämlich für nicht interaktive Shells gelöscht:

```
if [-z "$PS1"]; then
... Kommandos für nicht-interaktive Shell ...
else
... Kommandos für interaktive Shell ...
fi
```

Für eine interaktive Shell kann z.B. das Lesen von `~/.bashrc` mit **-norc** unterbunden oder aber mit **-rcfile** *dateiname* das Lesen einer anderen Datei als `~/.bashrc` erreicht werden. Um zu verhindern, daß die **bash** `~/.bash_profile` liest, müßte **-noprofile** gesetzt werden.

### Nicht interaktive Shells

Beim Anmelden gilt:

```
if Variable ENV nicht-NULL ist
then Lies Inhalt der Datei, deren Name in ENV angegeben, und führe ihn aus
```

Wenn die **bash** nicht im POSIX-Modus (Option **-posix** nicht angegeben) gestartet wird, wird **BASH\_ENV** vor **ENV** ausgewertet.

## 7.19 Abarbeitung von Kommandozeilen

Die Abarbeitung der Kommandozeilen entspricht bei der **bash** im wesentlichen der der Korn-Shell (siehe auch Kapitel 5.25)

## 7.20 Anwendungsbeispiel

Hier wird das bereits in Kapitel 5.28.1 vorgestellte Spiel moo, wo es als **ksh**-Skript realisiert wurde, als **bash**-Skript implementiert.

```
#-----
Spiel moo: aehneln dem bekannten Mastermind-Spiel.
moo generiert eine 4-stellige Zufallszahl,
welche der Spieler dann ermitteln muss.
Nach jedem Versuch gibt moo folgendes aus:
m Bullen; n Kuehe
m = Anzahl von richtigen Ziffern
an richtigen Positionen
n = Anzahl von richtigen Ziffern
an falschen Positionen
#-----

#--- Signal-Handler einrichten ---
#-----
trap 'echo -n "gesuchte Zahl war "
 for i in 1 2 3 4
 do echo -n ${ziffer_$i}
 done; echo; exit' HUP INT QUIT TERM

#--- Variablen-Deklarationen ---
#-----
declare -i bullen=0 kuehe=0 # Zaehlvariablen
declare -i versuche=0 # zaehlt die Versuche
declare -i ziffer_{1,2,3,4} # 4-stellige Zufallszahl
declare -i bziffer_{1,2,3,4} # vorgeschlagene Benutzer-Zahl
declare -i i j # Laufvariablen
declare -i zahl

#--- Funktions-Deklarationen ---
#-----
#.....
generiert eine 4-stellige Zufallszahl ins Array ziffer,
wobei jede Ziffer eindeutig ist.
erlaubte Zahlen sind z.B. 0398, 1526, 8463, usw.
nicht erlaubte Zahlen sind z.B. 1231, 1022, 8988, usw.
#.....
function bestimme_zufalls_zahl
{
 declare -i i j

 ziffer_1=RANDOM%10
 for i in 2 3 4
 do eval ziffer_$i=RANDOM%10
 j=1
 while [$j -lt $i]
```

```

 do
 if [${ziffer_$i} -eq ${ziffer_$j}]
 then eval ziffer_$i=RANDOM%10
 j=1
 else
 j=j+1
 fi
 done
 done
}

#.....
Eingebene Benutzerzahl bezueglich
Treffern (bullen und kuehe) analysieren
#.....
function zahl_analyse
{
 declare -i i j
 declare -i z_{1,2,3,4}

 bullen=0 kuehe=0
 for i in 1 2 3 4
 do eval z_$i=\${ziffer_$i}
 done

 for i in 1 2 3 4 # bullen zaehlen
 do if [${z_$i} -eq ${bziffer_$i}]
 then bullen=bullen+1
 eval z_$i=10
 fi
 done
 kuehe=0
 for i in 1 2 3 4
 do for j in 1 2 3 4
 do if [(($z_$i == bziffer_$j)) -eq 1]
 then kuehe=kuehe+1
 break
 fi
 done
 done
 echo ".....$bullen Bullen; $kuehe Kuehe"
}

#--- main -----
#-----

#... Zufallszahl bestimmen
bestimme_zufalls_zahl

until [$bullen -eq 4]
do

```

```
#... Zahl einlesen und pruefen
echo -n "Dein Vorschlag: "
read zahl
if [$[zahl >= 1 && zahl <= 9999] -eq 1]
then versuche=versuche+1
else echo "ungueltige Zahl" >&2
 continue
fi
#... Zahl in bziffer speichern
#... und analysieren
for i in 4 3 2 1
do eval bziffer_$i=zahl%10
 zahl=zahl/10
done
zahl_analyse
done
echo -e "\nDu hast die Zahl nach $versuche Versuchen gefunden"
#-----
```





# 8 tcsh – Erweiterte C-Shell

*Gott hat selber kein Kopftuch,  
doch nimmt er's der einen und  
bindet's der andern um.*

*Spanisches Sprichwort*

Die **tcsh** ist ein Nachfolger der C-Shell mit zahlreichen Erweiterungen. Sie ist als Freeware erhältlich und wird unter Linux standardmäßig mitgeliefert. Die tcsh verhält sich genauso wie die C-Shell, verfügt aber über eine Vielzahl von Erweiterungen oder Neuheiten, die hier vertieft besprochen werden. Grundlegende Konzepte, die völlig identisch zur C-Shell sind, werden nur kurz angesprochen. Detailliertere Informationen zu den entsprechenden Konstrukten können in Kapitel 6 nachgeschlagen werden, wo die C-Shell ausführlich beschrieben ist.

## 8.1 Aufrufsyntax der tcsh

**tcsh** [-bcdefFimnqstvVxX] [-Dname[=wert]] [argument(e)]

**tcsh -l**

Wenn das erste Argument (Argument 0) ein Querstrich (-) ist, dann wird eine Login-Shell gestartet. Eine Login-Shell kann auch mit **tcsh -l** gestartet werden.

Die Optionen bedeuten dabei im einzelnen folgendes:

| Option               | Bedeutung                                                                                                                        |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>-b</b>            | <b>(break)</b> Restliche Argumente werden nicht als Optionen interpretiert, selbst wenn sie mit Querstrich (-) beginnen.         |
| <b>-c text</b>       | <b>(command)</b> text legt auszuführende Kommandos fest. Sind weitere Argumente angegeben, werden diese in <b>argv</b> abgelegt. |
| <b>-d</b>            | <b>(directory stack)</b> Die tcsh lädt den Directorystack von <code>~/.cshdirs</code> (siehe auch Kapitel 8.2).                  |
| <b>-Dname[=wert]</b> | <b>(Define)</b> setzt die Environment-Variable <i>name</i> auf <i>wert</i> . Diese Option wird nur auf Domain / OS angeboten.    |
| <b>-e</b>            | <b>(exit)</b> tcsh verlassen, wenn ein Kommando nicht erfolgreich ausgeführt werden kann.                                        |
| <b>-f</b>            | <b>(fast)</b> tcsh führt <code>~/.tcshrc</code> nicht aus und startet deshalb schneller.                                         |

| Option                  | Bedeutung                                                                                                                                                                                           |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-F</b>               | ( <b>Fork</b> ) Die tcsh benutzt <i>fork</i> anstelle von <i>vfork</i> , um neue Prozesse zu starten. Diese Option wird nur auf Convex/OS angeboten.                                                |
| <b>-i</b>               | ( <b>interactive</b> ) als interaktive tcsh starten.                                                                                                                                                |
| <b>-l</b>               | ( <b>login</b> ) tcsh wird als Login-Shell gestartet. Neben <b>-l</b> dürfen keine weiteren Optionen angegeben sein.                                                                                |
| <b>-m</b>               | Die tcsh führt <i>~/tcshrc</i> sogar dann aus, wenn diese Datei nicht dem effektiven Benutzer gehört.                                                                                               |
| <b>-n</b>               | ( <b>noexec</b> ) Kommandos nur lesen und auf Syntaxfehler prüfen, aber nicht ausführen.                                                                                                            |
| <b>-s [argument(e)]</b> | Interaktive Subshell starten, die Kommandos von stdin liest. Eventuelle Argumente werden als Positionsparameter übergeben.                                                                          |
| <b>-t</b>               | Nur eine Kommandozeile lesen und ausführen. Ein Backslash (\) kann verwendet werden, um Sonderbedeutung eines Neuezeile-Zeichen auszuschalten, wenn Eingabe in einer Zeile fortgesetzt werden soll. |
| <b>-v</b>               | ( <b>verbose</b> ) setzt die Shellvariable <b>verbose</b> , so daß Kommandozeilen noch einmal nach der History-Substitution angezeigt werden.                                                       |
| <b>-V</b>               | ( <b>Verbose</b> ) setzt die Shellvariable <b>verbose</b> vor der Ausführung von <i>~/tcshrc</i> (siehe auch <b>-v</b> ).                                                                           |
| <b>-x</b>               | ( <b>xtrace</b> ) setzt die Shellvariable <b>echo</b> , so daß Kommandozeilen noch einmal unmittelbar vor ihrer Ausführung angezeigt werden.                                                        |
| <b>-X</b>               | ( <b>Xtrace</b> ) setzt die Shellvariable <b>echo</b> vor der Ausführung von <i>~/tcshrc</i> (siehe auch <b>-x</b> ).                                                                               |
| <b>-q</b>               | ( <b>sigquit</b> ) tcsh akzeptiert das Signal SIGQUIT; Jobkontrolle wird ausgeschaltet.                                                                                                             |

## 8.2 Startup-Dateien der tcsh

Hier zeigt die tcsh unterschiedliches Verhalten für Login-Shells und Shells, die keine Login-Shells sind.

### Login-Shell

Eine Login-Shell führt die Startup-Dateien in folgender Reihenfolge aus:

*/etc/csh.cshrc*

*/etc/csh.login*

*~/tcshrc* (bzw. *~/cshrc*, wenn *~/tcshrc* nicht existiert)

*~/history* (bzw. Datei, die in Variable **histfile** angegeben ist)

`~/.login`

`~/.cshdirs` (bzw. Datei, die in Variable **dirsfile** angegeben ist)

Eventuell führt die tcsh auch `/etc/csh.login` vor `/etc/csh.cshrc`, und `~/.login` vor `~/.tcshrc` (`~/.cshrc`) und `~/.history`. Dies ist abhängig davon, wie die tcsh kompiliert wurde (siehe auch Variable **version** in Kapitel 8.13).

### Shell, die keine Login-Shell ist

Solche Shells führen nur die beiden folgenden Startup-Dateien aus:

`/etc/csh.cshrc`

`~/.tcshrc` (bzw. `~/.cshrc`, wenn `~/.tcshrc` nicht existiert)

### Abmelden

Das Abmelden von einer tcsh kann über *Strg-D* in einer leeren Zeile, mit den Kommandos **logout** oder **login** oder über den *autologout*-Mechanismus der tcsh (siehe auch Variable **autologout** in Kapitel 8.13) erfolgen.

Wenn sich eine Login-Shell beendet, setzt sie zunächst die Variable **logout** entweder auf *normal* oder *automatic*, bevor sie die Kommandos in den Dateien `/etc/csh.logout` und `~/.logout` ausführt.

## 8.3 Editieren von Kommandozeilen

Hier wird zunächst *Grundsätzliches zum Kommandozeilen-Editor* beschrieben, bevor auf die *Automatische Vervollständigung und Korrektur von Wörtern* eingegangen wird. Abschließend werden alle möglichen *Editor-Kommandos* mit ihren vor-eingestellten Tastenkombinationen aufgelistet.

In diesem Abschnitt wird die emacs-Notation benutzt: Strg-Tasten werden mit **^** und Meta-Tasten werden mit **M-** angegeben. So bedeutet z.B. die Angabe **^x**, daß die Taste **x** bei gedrückter Strg-Taste zu drücken ist. Auf Tastaturen ohne Meta-Taste (*Alt*-Taste) entspricht **M-** dem Drücken der *ESC*-Taste. So bedeutet z.B. die Angabe **M-w**, daß zunächst die *ESC*-Taste und dann die Taste **w** zu drücken ist bzw. daß bei gedrückter *Alt*-Taste die Taste **w** zu drücken ist. **M-^x** bedeutet dann, daß zunächst die *ESC*-Taste und dann die Taste **w** bei gedrückter Strg-Taste einzugeben ist bzw. daß bei gleichzeitigem Drücken der *Alt*- und Strg-Taste noch die Taste **w** zu drücken ist.

### 8.3.1 Grundsätzliches zum Kommandozeilen-Editor

Das Editieren von Kommandozeilen ist nur dann möglich, wenn die Variable **edit** gesetzt ist. Beim Start von interaktiven Shells wird diese Variable automatisch gesetzt.

Für das Editieren von Kommandozeilen stehen Editor-Kommandos zur Verfügung, die weitgehend denen von **emacs** entsprechen. Ein **vi**-ähnlicher Editor kann jedoch auch eingestellt werden.

Mit dem neuen builtin-Kommando **bindkey** (siehe Kapitel 8.11) kann der Benutzer eigene Tastenkombinationen für bestimmte Editor-Kommandos festlegen, oder die momentan eingestellten Tasten für einzelne oder alle Editor-Kommandos erfragen.

Die vier Pfeiltasten werden immer automatisch eingestellt:

*Pfeil-oben*     (**up-history**) vorherige Zeile aus History-Liste holen

*Pfeil-unten*   (**down-history**) nächste Zeile aus History-Liste holen

*Pfeil-links*    (**backward-char**) Cursor ein Zeichen nach links

*Pfeil-rechts*   (**forward-char**) Cursor ein Zeichen nach rechts

Alle aktuell eingestellten Tastenkombinationen (*key-bindings*) können mit dem einfachen Aufruf

### **bindkey**

aufgelistet werden.

Wortgrenzen sind für den Kommandozeilen-Editor anders festgelegt als für die *tcsh*. Während die *tcsh* nämlich nur Leer-, Tab- und einige wenige Sonderzeichen als Wortende interpretiert, legen beim Kommandozeilen-Editor alle Nicht-Buchstaben und Nicht-Ziffern, die nicht in der Variable **wordchars** angegeben sind, Wortgrenzen fest.

## 8.3.2 Automatische Vervollständigungen

### **Das Editor-Kommando *complete-word***

Der Kommandozeilen-Editor versucht bei Eingabe des Editor-Kommandos *complete-word* (auf die Tabulator-Taste voreingestellt) die Vervollständigung des aktuellen Wortes. Ist eine solche Vervollständigung möglich, werden die entsprechenden fehlenden Zeichen eingefügt.

Kann dabei ein Wort nicht vollkommen vervollständigt werden, weil es mehrere mögliche Vervollständigungen gibt, werden nur die entsprechenden Zeichen eingefügt und der Benutzer durch die Terminalglocke darauf hingewiesen, daß das Wort noch unvollständig ist.

Konnte ein Wort vollkommen vervollständigt werden, wird am Ende entweder ein Slash /, wenn es sich bei diesem Wort um ein Directory handelt, oder sonst ein Leerzeichen angehängt. Um dieses automatische Anhängen eines Slashes / oder eines Leerzeichens auszuschalten, muß die Variable **addsuffix** gelöscht werden.

Vervollständigung eines Wortes ist nicht nur am Ende, sondern auch in der Mitte einer Zeile möglich, wobei durch das Einfügen der entsprechenden Zeichen der Rest der Zeile nach rechts verschoben wird.

Kommandos und Variablen können ebenfalls automatisch durch Drücken der Tab-Taste vervollständigt werden. Gibt man z.B. folgendes ein

```
> base[TAB-Taste]
```

dann wird diese Eingabe zu

```
> basename _ (_=Cursor; es wurde automatisch ein Leerzeichen eingefügt)
```

vervollständigt, wenn **basename** das einzige Kommando ist, das mit *base* beginnt. In diesem Fall durchsucht der Kommandozeilen-Editor alle in **path** angegebenen Directories, bzw. wenn es sich bei dem Wort um einen Pfadnamen handelt im entsprechenden Pfad, nach Kommandos mit dem bisher eingegebenen Präfix. Gibt man z.B. folgendes ein

```
> echo $ar[TAB-Taste]
```

dann wird diese Eingabe zu

```
> echo $argv
```

vervollständigt, wenn keine andere Variable existiert, die mit *ar* beginnt.

Die tcsh wendet folgende Regeln an, um zu entscheiden, ob ein Wort ein Kommando, eine Variable oder ein Dateiname ist:

- ▶ Beginnt das Wort mit einem **\$**, versucht sie es zu einem Variablennamen zu vervollständigen.
- ▶ Ist das Wort das erste Wort in der Zeile oder folgt es einem **;**, **|**, **|&**, **&&** oder **| |**, versucht sie es zu einem Kommandonamen zu vervollständigen.
- ▶ In allen anderen Fällen versucht sie es zu einem Dateinamen zu vervollständigen.

### Das Editor-Kommando *delete-char-or-list*

Der Kommandozeilen-Editor listet bei Eingabe des Editor-Kommandos *delete-char-or-list* (auf **^D** voreingestellt) mittels **ls -F** alle möglichen Vervollständigungen des aktuellen Wortes, wie z.B.

```
> ls /usr/[^D]
lib/ local/ lost+found/
> ls /usr/
```

**b[^D]** listet z.B. alle Kommandonamen auf, die über **PATH** gefunden werden und mit **b** beginnen.

### Die tcsh-Variable autolist

Ist die Variable **autolist** gesetzt, werden bei einem Wort, das nicht vollkommen vervollständigt werden kann, alle Vervollständigungsmöglichkeiten ausgegeben:

```
> set autolist
> ls /usr/bin/v[TAB-Taste]
vi@ view@ vim*
> ls /usr/bin/vi (i wurde zu v hinzugefügt)
```

Wird die Variable **autolist** auf *ambiguous* gesetzt, werden die Vervollständigungsmöglichkeiten nur dann angezeigt, wenn die Vervollständigung fehlschlägt und keine neuen Zeichen dem Wort hinzugefügt wurden.

```
> set autolist=ambiguous
> ls /usr/bin/v[TAB-Taste]
> ls /usr/bin/vi[TAB-Taste]
vi@ view@ vim*
> ls /usr/bin/vi
```

### Variablen, Home-Directories und Directorystack

Ein zu vervollständigender Dateiname kann Variablen, ~ (Tilde für Home-Directory) oder = (für Einträge aus dem Directorystack; siehe auch 8.5.4) enthalten. z.B.

```
> ls ~h[^D]
halt hans hh holler hunter
> ls ~hu[TAB-Taste]
> ls ~hunter
```

oder

```
> set ubin = /usr/bin
> ls $ub[TAB-Taste]
> ls $ubin/v[^D]
vi@ view@ vim*
> ls $ubin/v
```

Um sich z.B. alle Login-Namen am System auflisten zu lassen, muß folgendes eingegeben werden:

```
~[^D]
```

Das Einsetzen eines Wertes anstelle eines Variablennamens ist mit dem Editor-Kommando *expand-variables* (voreingestellt auf *^X-\$*) möglich, wie z.B.

```
> set ubin = /usr/bin
> ls $ubin[^X-$]
> ls /usr/bin
```

**Die Editor-Kommandos *delete-char-or-list-or-eof* und *list-choices***

Das Editor-Kommando *delete-char-or-list-or-eof* (voreingestellt auf **^D**) hat abhängig von der Cursorposition mehrere Bedeutungen:

- ▶ Am Zeilenende listet es alle Vervollständigungsmöglichkeiten auf (siehe auch vorher).
- ▶ In der Zeilenmitte löscht es das Zeichen unter dem Cursor.
- ▶ In einer Leerzeile beendet es die tcsh, wenn die Variable **ignoreeof** nicht gesetzt ist.

Das Editor-Kommando *list-choices* (voreingestellt auf **M-^D**) listet unabhängig von der Cursorposition immer alle Vervollständigungsmöglichkeiten für das Wort vor dem Cursor auf.

**Die Editor-Kommandos *complete-word-fwd* und *complete-word-back***

Mit den beiden Editor-Kommandos *complete-word-fwd* und *complete-word-back*, die beide nicht mit Tasten vorbelegt sind, ist es möglich, alle Vervollständigungsmöglichkeiten schrittweise vor- oder rückwärts zu durchlaufen.

```
> bindkey ^F complete-word-fwd
> bindkey ^R complete-word-back
> ls /usr/bin/vi[TAB-Taste]
vi@ view@ vim*
> ls /usr/bin/vi[^F]
> ls /usr/bin/vi [^F]
> ls /usr/bin/view [^F]
> ls /usr/bin/vim [^R]
> ls /usr/bin/view
```

**Die tcsh-Variable *ignore***

Die in der Variable **ignore** enthaltenen Suffizes werden bei Wortvervollständigungen ignoriert, wie z.B.

```
> ls
add.c sub.t.c mult.c divi.c
add.c~ sub.t.c~ mult.c~ divi.c~
add.o sub.t.o mult.o divi.o
> set ignore = (.o \~)
> vi ad[^D]
add.c add.c~ add.o
> vi ad[TAB-Taste]
> vi add.c
```

Wie aus dem Beispiel ersichtlich wird, werden die Dateinamen, die ein Suffix aus **ignore** haben, zwar bei der Wortvervollständigung, aber nicht beim Auflisten der Vervollständigungsmöglichkeiten ignoriert.



### Die **tcsh**-Variable **complete**

Ist die Variable **complete** auf *enhance* gesetzt, wird

- ▶ nicht mehr zwischen Groß- und Kleinschreibung unterschieden
- ▶ die Zeichen: Punkt (.), Querstrich (-) und Unterstrich (\_) als Wortende interpretiert.
- ▶ die beiden Zeichen: Querstrich (-) und Unterstrich (\_) nicht mehr unterschieden.

### Die **tcsh**-Variable **recexact**

Ist die Variable **recexact** gesetzt, wird die kürzest mögliche Wortvervollständigung akzeptiert, selbst wenn längere Vervollständigungen möglich wären, wie z.B.

```
> ls
wald waldblume waldmeister walker
> set recexact
> ls wal[TAB-Taste]
```

erzeugt lediglich einen Piepston, da *wal* mehrdeutig ist. Wenn wir aber noch ein *d* eingeben

```
> ls wald[TAB-Taste]
> ls wald _
```

gilt die Wortvervollständigung als abgeschlossen, obwohl weitere Abdeckungen: wie *waldblume* und *waldmeister* möglich wären.

### Weitere **tcsh**-Variablen, die die Vervollständigung beeinflussen

- ▶ Ist Variable **autoexpand** gesetzt, wird das Editor-Kommando *expand-history* vor der Vervollständigung ausgeführt.
- ▶ Ist Variable **autocorrect** gesetzt, wird eine automatische Tippfehler-Korrektur (siehe auch Kapitel 8.3.3) durchgeführt, bevor eine Vervollständigung versucht wird.
- ▶ Ist Variable **correct** gesetzt, wird automatisch eine Vervollständigung versucht, nachdem *Return* eingegeben wurde.
- ▶ Die Variable **matchbeep** kann mit verschiedenen Werten gesetzt werden, die festlegen, ob in bestimmten Situationen von Vervollständigungen der Piepston auszugeben ist oder nicht.
- ▶ Ist Variable **nobeep** gesetzt, wird niemals der Piepston ausgegeben.

- ▶ Über die beiden Variablen **listmax** und **listmaxrows** kann ein Limit für die Vervollständigungsmöglichkeiten festgelegt werden, ab dem der Benutzer gefragt wird, ob diese wirklich aufzulisten sind. **listmax** legt dabei das Limit für die Namen insgesamt und **listmaxrows** für die Zeilen fest.
- ▶ Ist Variable **recognize\_only\_executables** gesetzt, werden bei Vervollständigungsmöglichkeiten nur ausführbare Dateien aufgelistet.

### Das builtin-Kommando complete

Mit dem builtin-Kommando **complete** kann festgelegt werden wie Worte, die keine Dateinamen, Variablennamen oder Kommandos sind, zu vervollständigen sind (siehe auch Kapitel 8.11).

## 8.3.3 Automatische Tippfehler-Korrektur

Manchmal ist die tcsh in der Lage, falsch getippte Kommando-, Variablen- oder Dateinamen zu erkennen und automatisch zu korrigieren.

Einzelne Wörter können mit dem Editor-Kommando *spell-word* (voreingestellt auf **M-S**) korrigiert werden. Um in der ganzen Zeile die Tippfehler zu korrigieren, steht das Editor-Kommando *spell-line* (voreingestellt auf **M-\$**) zur Verfügung.

```
> la /urs/bin[M-$]
> ls /usr/bin
```

Es ist auch möglich, über die Variable **correct** festzulegen, was zu korrigieren ist. Hat **correct** den Wert *cmd*, wird nur der Kommandoname bei der Übergabe der Kommandozeile mit *Return* korrigiert. Ist **correct** auf *all* gesetzt, wird bei *Return* die ganze Kommandozeile korrigiert.

```
> set correct = all
> la /urs
CORRECT>ls /usr (y|n|e|a)?
```

Entdeckt die tcsh Tippfehler in der Zeile, korrigiert sie diese, zeigt dem Benutzer vor der Ausführung die entsprechend korrigierte Zeile an und läßt ihn nun auswählen, was zu tun ist: Gibt er **y** oder *Leerzeichen* ein, wird die korrigierte Zeile ausgeführt. Gibt der Benutzer **n** oder *Return* ein, wird die ursprüngliche und unkorrigierte Zeile ausgeführt. Gibt der Benutzer **e** ein, wird nichts zur Ausführung gebracht und die ursprüngliche und unkorrigierte Zeile wieder zur weiteren Editierung ausgegeben. Gibt der Benutzer **a** ein, wird die Verarbeitung der Kommandozeile abgebrochen, als ob **^C** gedrückt worden wäre. Ist die Variable **autocorrect** gesetzt, wird automatisch vor jedem Vervollständigungsversuch eines Wortes dieses zuvor korrigiert.

### 8.3.4 Editor-Kommandos

Mit dem neuen builtin-Kommando **bindkey** (siehe Kapitel 8.11) kann der Benutzer eigene Tastenkombinationen für bestimmte Editor-Kommandos festlegen oder die momentan eingestellten Tasten für einzelne oder alle Editor-Kommandos erfragen.

#### **bindkey -l**

listet die Editor-Kommandos mit einer kurzen Beschreibung auf. Hier werden nicht alle, wohl aber die wichtigsten Editor-Kommandos beschrieben. In den entsprechenden Man-pages zur *tcsh*, *emacs(1)* und *vi(1)* können eventuell weitere benötigte Editor-Kommandos nachgeschlagen werden.

### 8.3.5 emacs-Editor-Kommandos

Um die emacs-Editor-Kommandos einzustellen, ist

#### **bindkey -e**

aufzurufen. Meist ist *emacs* schon voreingestellt, so daß dieser Aufruf nicht notwendig ist.

Die für die emacs-Editor-Kommandonamen voreingestellten Tastenkombinationen (*key-bindings*) werden in der folgenden Übersicht zuerst und die entsprechenden Editor-Kommandonamen in Klammern angegeben. Ein Querstrich (-) zeigt im folgenden immer an, daß für das betreffende Editor-Kommando keine Tastenkombination voreingestellt ist.

| <i>Cursorpositionierung</i>     |                                                                        |
|---------------------------------|------------------------------------------------------------------------|
| <b>^F</b> , <i>Pfeil-rechts</i> | <i>(forward-char)</i> ein Zeichen nach rechts                          |
| <b>^B</b> , <i>Pfeil-links</i>  | <i>(backward-char)</i> ein Zeichen nach links                          |
| <b>^A</b>                       | <i>(beginning-of-line)</i> an Zeilenanfang                             |
| <b>^E</b>                       | <i>(end-of-line)</i> ans Zeilenende                                    |
| <b>M-F</b>                      | <i>(forward-word)</i> hinter aktuelles bzw. nächstes Wort              |
| <b>M-B</b>                      | <i>(backward-word)</i> auf Anfang des aktuellen bzw. vorherigen Wortes |
| <b>M-n ^F</b>                   | <i>n</i> Zeichen nach rechts                                           |
| <b>M-n ^B</b>                   | <i>n</i> Zeichen nach links                                            |
| <b>M-n M-F</b>                  | nach rechts hinter das <i>n</i> .te nächste Wort                       |
| <b>M-n M-B</b>                  | nach links auf Anfang des <i>n</i> .ten vorherigen Wortes              |

| Löschen                             |                                                                                                                                                                                     |
|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Backspace</i> , <b>^H</b>        | ( <i>backward-delete-char</i> ) Zeichen links vom Cursor löschen                                                                                                                    |
| <b>^D</b>                           | ( <i>delete-char-or-list-or-eof</i> ) Zeichen an Cursorposition löschen ( <i>delete-char</i> ) oder <i>list-choices</i> am Zeilenende oder <i>end-of-file</i> in einer leeren Zeile |
| <b>^U</b>                           | ( <i>kill-whole-line</i> ) ganze Zeile löschen und in Löschpuffer aufheben                                                                                                          |
| <b>^K</b>                           | ( <i>kill-line</i> ) Ab Cursor bis Zeilenende löschen und in Löschpuffer aufheben                                                                                                   |
| <b>M-D</b>                          | ( <i>delete-word</i> ) Ab Cursor bis Wortende löschen und in Löschpuffer aufheben                                                                                                   |
| <b>M-Backspace</b> ,<br><b>M-^H</b> | ( <i>backward-delete-word</i> ) Ab Cursor (nach links) bis Wortanfang löschen und in Löschpuffer aufheben                                                                           |
| <b>^W</b>                           | ( <i>kill-region</i> ) Ab Cursor bis zu einer Marke löschen und in Löschpuffer aufheben                                                                                             |
| <b>M-n Backspace</b>                | <i>n</i> Zeichen links vom Cursor löschen                                                                                                                                           |
| <b>M-n ^H</b>                       | <i>n</i> Zeichen links vom Cursor löschen                                                                                                                                           |
| <b>M-n ^D</b>                       | <i>n</i> Zeichen ab Cursor (nach rechts) löschen                                                                                                                                    |
| <b>M-n ^U</b>                       | Ab Cursor bis Zeilenanfang löschen                                                                                                                                                  |
| <b>M-n ^D</b>                       | ab Cursor (nach rechts) <i>n</i> Wörter löschen                                                                                                                                     |
| <b>M-n M-Backspace</b>              | ab Cursor (nach links) <i>n</i> Wörter löschen und in Löschpuffer aufheben                                                                                                          |
| <b>M-n M-^H</b>                     | ab Cursor (nach links) <i>n</i> Wörter löschen und in Löschpuffer aufheben                                                                                                          |
| –                                   | ( <i>delete-char</i> ) Zeichen an Cursorposition löschen; siehe auch <i>delete-char-or-list-or-eof</i>                                                                              |
| –                                   | ( <i>delete-char-or-eof</i> ) Zeichen an Cursorposition löschen ( <i>delete-char</i> ) oder <i>end-of-file</i> in einer leeren Zeile; siehe auch <i>delete-char-or-list-or-eof</i>  |
| –                                   | ( <i>delete-char-or-list</i> ) Zeichen an Cursorposition löschen ( <i>delete-char</i> ) oder <i>list-choices</i> am Zeilenende; siehe auch <i>delete-char-or-list-or-eof</i>        |
| –                                   | ( <i>backward-kill-line</i> ) von Cursorposition bis Zeilenanfang löschen und in Löschpuffer aufheben                                                                               |
| Editieren von History-Kommandos     |                                                                                                                                                                                     |
| <b>^P</b> , Pfeil-oben              | ( <i>up-history</i> ) vorherige Kommandozeile aus History-Liste holen                                                                                                               |
| <b>^N</b> , Pfeil-unten             | ( <i>down-history</i> ) nächste Kommandozeile aus History-Liste holen                                                                                                               |

|                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>M-P</b>                          | ( <i>history-search-backward</i> ) Rückwärts in History-Liste nach Zeile suchen, die mit bisher eingegebenem String beginnt. Bei jedem neuen <b>M-P</b> wird sofort die nächste vorherige Zeile angezeigt, die mit dem gleichen String beginnt.                                                                                                                                                                                                                                                                                                                                                |
| <b>M-N</b>                          | ( <i>history-search-forward</i> ) wie <i>history-search-backward</i> , nur vorwärts in History-Liste suchen                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| -                                   | ( <i>i-search-back</i> ) Es wird zunächst der Prompt "bck:" ausgegeben. Werden nun Zeichen eingegeben, wird immer sofort nach einer weiteren Zeile mit dem neuen Text gesucht. Mit <i>Backspace</i> ( <i>delete-char</i> ) können Zeichen aus dem Suchmuster gelöscht werden. Mit <b>^W</b> kann das aktuelle Wort dem Suchmuster hinzugefügt werden. <i>ESC</i> bricht genauso wie jedes andere Zeichen, das nicht dem <i>self-insert-command</i> zugeteilt ist, die Suche ab. Wenn die Suche nicht erfolgreich war, kann mit <b>^G</b> zur letzten erfolgreichen Suche zurückgekehrt werden. |
| -                                   | ( <i>i-search-fwd</i> ) wie <i>i-search-back</i> , nur daß die Suche vorwärts ("fwd:") durchgeführt wird                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>M-n ^P, M-n<br/>^Pfeil-oben</b>  | <i>n</i> .te vorherige Kommandozeile aus History-Liste holen                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>M-n ^N, M-n<br/>^Pfeil-unten</b> | <i>n</i> .te nächste Kommandozeile aus History-Liste holen                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

#### Automatische Vervollständigungen

|                |                                                                                                                                                                                                                                                                             |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Tab, ^I</b> | ( <i>complete-word</i> ) vervollständigt ein Wort wie in Kapitel 8.3.2 beschrieben                                                                                                                                                                                          |
| -              | ( <i>complete-word-fwd</i> ) alle Vervollständigungsmöglichkeiten schrittweise vorwärts durchlaufen; siehe auch Kapitel 8.3.2                                                                                                                                               |
| -              | ( <i>complete-word-back</i> ) alle Vervollständigungsmöglichkeiten schrittweise rückwärts durchlaufen; siehe auch Kapitel 8.3.2                                                                                                                                             |
| <b>^X-Tab</b>  | ( <i>complete-word-raw</i> ) wie <i>complete-word</i> , nur daß benutzerdefinierte Vervollständigungsoptionen ignoriert werden                                                                                                                                              |
| <b>^X-\$</b>   | ( <i>expand-variables</i> ) vervollständigt den Variablennamen links vom Cursor; siehe auch Variable-Substitutionen                                                                                                                                                         |
| <b>^X-*</b>    | ( <i>expand-glob</i> ) expandiert das Expandierungszeichen vor dem Cursor, wie z.B.<br><pre>&gt; ls hans meier sccs_rcs schule src &gt; ls s*[^X-*]</pre> resultiert dann in<br><pre>&gt; ls sccs_rcs schule src</pre> Die folgende Eingabe<br><pre>&gt; cp *.c[^X-*]</pre> |

|                                                  |                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                                  | würde anstelle von <b>*.c</b> alle C-Programme des Working-Directories einsetzen.                                                                                                                                                                                                                                                                                                   |
| <b>^X-N</b>                                      | ( <i>normalize-path</i> ) expandiert das aktuelle Wort entsprechend den Vorgaben der Shellvariable <b>symlinks</b> ; siehe auch Kapitel 8.13.                                                                                                                                                                                                                                       |
| <b>^X-?</b>                                      | ( <i>normalize-command</i> ) sucht nach dem aktuellen Wort in den <b>PATH</b> -Directories. Ist das Wort gefunden, wird es durch den vollen Pfadnamen ersetzt. Dieses Kommando ist hilfreich bei Kommandos, die Kommandos als Argumente erfordern, wie z.B. <b>dbx</b> oder <b>sh -c</b> .                                                                                          |
| <b>M-/</b>                                       | ( <i>dapprev-expand</i> ) durchsucht rückwärts die Kommandozeile und History-Liste nach einem Wort, das mit dem aktuellen Wortstring beginnt. Findet es ein solches, setzt es dieses für das aktuelle Wort ein. Findet es keines, sucht es vom Ende der History-Liste rückwärts bis zur aktuellen Cursorposition.                                                                   |
| <b>M-R</b>                                       | ( <i>toggle-literal-history</i> ) expandiert History-Substitutionen bzw. macht durchgeführte History-Substitutionen wieder rückgängig.                                                                                                                                                                                                                                              |
| <b>M-Leertaste, M-!</b>                          | ( <i>expand-history</i> )<br><br>History-Substitution für aktuelles Wort durchführen; siehe auch History-Substitution und Variable <b>autoexpand</b>                                                                                                                                                                                                                                |
| -                                                | ( <i>magic-space</i> ) expandiert wie <i>expand-history</i> History-Substitutionen in der aktuellen Kommandozeile und hängt ein Leerzeichen an.                                                                                                                                                                                                                                     |
| -                                                | ( <i>expand-line</i> ) wie <i>expand-history</i> , nur daß History-Substitutionen für jedes Wort durchgeführt werden.                                                                                                                                                                                                                                                               |
| <i>Liste der Vervollständigungsmöglichkeiten</i> |                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>M-^D</b>                                      | ( <i>list-choices</i> ) listet (mit <i>ls -f</i> ) alle möglichen Vervollständigungen für das aktuelle Wort wie in Kapitel 8.3.2 beschrieben; siehe auch <i>delete-char-or-list-or-eof</i> und <i>list-choices-raw</i> . <b>M-^D</b> sollte in einer Leerzeile oder in der Mitte der Zeile verwendet werden, da hier <b>^D EOF</b> bzw. das Löschen eines Zeichens nach sich zieht. |
| <b>^X-^D</b>                                     | ( <i>list-choices-raw</i> ) wie <i>list-choices</i> , nur daß benutzerdefinierte Vervollständigungsoptionen ignoriert werden.                                                                                                                                                                                                                                                       |
| <b>^X-G</b>                                      | ( <i>list-glob</i> ) listet (mit <i>ls -F</i> ) alle Wörter auf, die durch das Expandierungszeichen im Wort vor dem Cursor abgedeckt sind. So listet z.B.<br><br><b>*.c[^X-G]</b><br><br>alle C-Programme des Working-Directories auf.                                                                                                                                              |
| -                                                | ( <i>list-or-eof</i> ) <i>list-choices</i> oder <i>end-of-file</i> in einer leeren Zeile; siehe auch <i>delete-char-or-list-or-eof</i>                                                                                                                                                                                                                                              |

| <i>Automatische Tippfehler-Korrektur</i> |                                                                                                                                                                                                                                                                    |
|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>M-\$</b>                              | ( <i>spell-line</i> ) führt eine automatische Tippfehler-Korrektur für die ganze Kommandozeile durch. Es werden dabei jedoch Wörter ignoriert, deren erstes Zeichen ein !, ., \, ^, * oder % ist, um eventuelle Probleme mit Substitutionen zu vermeiden.          |
| <b>M-S</b>                               | ( <i>spell-word</i> ) führt eine automatische Tippfehler-Korrektur für das aktuelle Wort durch. Beginnt das Wort jedoch mit einem der Zeichen ein !, ., \, ^, * oder %, wird keine Korrektur durchgeführt, um eventuelle Probleme mit Substitutionen zu vermeiden. |
| <i>Markieren und Kopieren</i>            |                                                                                                                                                                                                                                                                    |
| <b>^@</b>                                | ( <i>set-mark-command</i> ) Cursorposition markieren                                                                                                                                                                                                               |
| <b>^X-^X</b>                             | ( <i>exchange-point-and-mark</i> ) Cursorposition markieren und Cursor an zuvor markierter Stelle positionieren                                                                                                                                                    |
| <b>M-W</b>                               | ( <i>copy-region-as-kill</i> ) Text zwischen Cursor und einer zuvor markierten Stelle im Löschpuffer speichern.                                                                                                                                                    |
| <b>^Y</b>                                | ( <i>yank</i> ) Inhalt des Löschpuffers an Cursorposition kopieren                                                                                                                                                                                                 |
| <i>Sonstiges</i>                         |                                                                                                                                                                                                                                                                    |
| <b>^T</b>                                | ( <i>transpose-chars</i> ) Zeichen an Cursorposition mit vorherigem vertauschen                                                                                                                                                                                    |
| <b>-</b>                                 | ( <i>gosmacs-transpose-chars</i> ) Die beiden Zeichen vor Cursorposition vertauschen                                                                                                                                                                               |
| <b>M-U</b>                               | ( <i>upcase-word</i> ) Aktuelles (oder folgendes Wort) ab Cursor in Großbuchstaben umwandeln                                                                                                                                                                       |
| <b>M-L</b>                               | ( <i>downcase-word</i> ) Aktuelles (oder folgendes Wort) ab Cursor in Kleinbuchstaben umwandeln                                                                                                                                                                    |
| <b>M-C</b>                               | ( <i>capitalize-word</i> ) Bei aktuellen (oder folgenden) Wort Anfangsbuchstaben in Großbuchstaben und Rest in Kleinbuchstaben umwandeln                                                                                                                           |
| <b>^L, M-^L</b>                          | ( <i>clear-screen</i> ) Bildschirm löschen und aktuelle Zeile oben anzeigen                                                                                                                                                                                        |
| <b>^R</b>                                | ( <i>redisplay</i> ) Bildschirm löschen und erneut anzeigen                                                                                                                                                                                                        |
| <b>^J, ^M, Return</b>                    | ( <i>newline</i> ) Kommandozeile ausführen                                                                                                                                                                                                                         |
| <b>M-^_</b>                              | ( <i>copy-prev-word</i> ) An Cursorposition vorheriges bzw. aktuelles Wort einfügen                                                                                                                                                                                |
| <b>M-_</b>                               | ( <i>insert-last-word</i> ) An Cursorposition das letzte Argument (letztes Wort) der vorherigen Kommandozeile einfügen                                                                                                                                             |
| <b>^V</b>                                | ( <i>quoted-insert</i> ) Ausschalten des nachfolgenden Sonderzeichens                                                                                                                                                                                              |

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>M-^Z</b>              | ( <i>run-fg-editor</i> ) sichert aktuelle Kommandozeile und sucht dann nach einem angehaltenen Job, dessen Name gleich dem in der Environment-Variable <b>EDITOR</b> oder <b>VISUAL</b> angegebenen Namen ist. Ist weder <b>EDITOR</b> noch <b>VISUAL</b> gesetzt, wird nach einem angehaltenen Job mit dem Namen <i>ed</i> oder <i>vi</i> gesucht. Wird ein solcher Job gefunden, wird dieser Job mit <b>fg %jobname</b> im Vordergrund zur Ausführung gebracht. So kann schnell und leicht zwischen der Shell und dem Editor hin- und hergeschaltet werden. Oft wird <i>run-fg-editor</i> auf <b>^Z</b> eingestellt, um das ständige Hin- und Herschalten noch zu erleichtern. |
| <b>M-H</b>               | ( <i>run-help</i> ) durchsucht die Dokumentation nach dem aktuellen Kommando. Diese Dokumentation sollte kurz sein, da keine seitenweise Ausgabe hierfür zur Verfügung steht. Zudem sollte sie sich in Dateien mit Namen wie <i>kommando.help</i> , <i>kommando.1</i> , <i>kommando.6</i> , <i>kommando.8</i> oder <i>kommando</i> befinden. Diese Dateien sollten sich in einem Directory befinden, das in der Environment-Variablen <b>HPATH</b> angegeben ist. Es wird grundsätzlich immer nur die erste gefundene Datei ausgegeben.                                                                                                                                          |
| <b>M-?</b>               | ( <i>which-command</i> ) Das <b>which</b> -Kommando für das erste Wort der Kommandozeile ausführen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>-</b>                 | ( <i>overwrite-mode</i> ) zwischen den beiden Editier-Modi "Einfügen" und "Überschreiben" hin- und herschalten; siehe auch Variable <b>inputmode</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>a,b,c,...,1,2,...</b> | ( <i>self-insert-command</i> ) Alle diesem Kommando zugeteilten Zeichen werden bei Eingabe an der entsprechenden Stelle eingefügt, bzw. überschreiben das Cursorzeichen (siehe <i>overwrite-mode</i> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>^X, ^[</b>            | ( <i>sequence-lead-in</i> ) zeigt an, daß das folgende Zeichen eine Multizeichen-Sequenz ist.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>^G, ...</b>           | ( <i>undefined-key</i> ) alle Tastenkombinationen, die keinem Editor-Kommando zugeteilt sind. Wird eine solche Tastenkombination eingegeben, ertönt der Piepston.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>M-n M-U</b>           | <i>n</i> Worte ab Cursorposition in Großbuchstaben umwandeln                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>M-n M-L</b>           | <i>n</i> Worte ab Cursorposition in Kleinbuchstaben umwandeln                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>M-n M-C</b>           | Bei aktuellem (oder folgendem) Wort Anfangsbuchstabe in Großbuchstaben und restlichen <i>n</i> Worte in Kleinbuchstaben umwandeln                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>M-n M-^_</b>          | An Cursorposition letzte <i>n</i> Worte einfügen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>M-n M-_</b>           | An Cursorposition letzte <i>n</i> Worte der vorherigen Kommandozeile einfügen.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |



### 8.3.6 vi-Editor-Kommandos

Um die **vi**-Editor-Kommandos einzustellen, ist

#### **bindkey -v**

aufzurufen.

Die für die *vi*-Editor-Kommandonamen voreingestellten Tastenkombinationen (*key-bindings*) werden in der folgenden Übersicht zuerst und die entsprechenden Editor-Kommandonamen in Klammern angegeben. In der folgenden Übersicht sind nahezu alle möglichen *vi*-Kommandos angegeben.

#### **Umschalten in Einfüge- bzw. Kommandomodus**

Anders als im wirklichen **vi** ist der Einfügemodus bereits zu Beginn eingeschaltet.

- i** (*vi-insert*) Einfügemodus einschalten
- a** (*vi-add*) Einfügemodus einschalten, wobei Cursor um eine Position nach rechts bewegt wird
- I** (*vi-insert-at-bol*) Einfügemodus einschalten, wobei Cursor auf den Zeilenanfang positioniert wird
- A** (*vi-add-at-eol*) Einfügemodus einschalten, wobei Cursor am Zeilenende positioniert wird
- R** (*vi-replace-mode*) an Cursorposition in Einfügemodus (besser: Überschreibmodus) umschalten, wobei hier nicht eingefügt wird, sondern alle nachfolgend eingegebenen Zeichen eventuell schon vorhandene Zeichen überschreiben
- C** (*change-till-end-of-line*) ab Cursorposition bis Zeilenende alle Zeichen löschen und in Einfügemodus umschalten
- S** (*vi-substitute-line*) ganze Zeile löschen und in Einfügemodus umschalten
- s** (*vi-substitute-char*) Zeichen an Cursorposition löschen und in Einfügemodus umschalten
- ESC** (*vi-cmd-mode*) Vom Einfügemodus in den Kommandomodus zurückschalten

| <i>Cursorpositionierung</i>                            |                                                                                                                                                           |
|--------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>l</b> , <i>Pfeil-rechts</i> ,<br><i>Leerzeichen</i> | ( <i>forward-char</i> ) ein Zeichen nach rechts; im Einfügemodus <b>^F</b>                                                                                |
| <b>h</b> , <i>Pfeil-links</i>                          | ( <i>backward-char</i> ) ein Zeichen nach links; im Einfügemodus <b>^B</b>                                                                                |
| <b>^A</b> , <b>0</b>                                   | ( <i>beginning-of-line</i> ) an Zeilenanfang; <b>0</b> nur im Kommandomodus                                                                               |
| <b>^E</b> , <b>\$</b>                                  | ( <i>end-of-line</i> ) ans Zeilenende; <b>\$</b> nur im Kommandomodus                                                                                     |
| <b>w</b>                                               | ( <i>vi-beginning-of-next-word</i> ) auf Anfang des nächsten Wortes; Sonderzeichen wie Kommata zählen dabei als eigene Wörter; nur im Kommandomodus       |
| <b>W</b>                                               | ( <i>vi-word-fwd</i> ) auf Anfang des nächsten Wortes; Sonderzeichen wie Kommata zählen dabei nicht als eigene Wörter; nur im Kommandomodus               |
| <b>e</b>                                               | ( <i>vi-eword</i> ) ans Ende des aktuellen bzw. nächsten Wortes; Sonderzeichen wie Kommata zählen dabei als eigene Wörter; nur im Kommandomodus           |
| <b>E</b>                                               | ( <i>vi-endword</i> ) ans Ende des aktuellen bzw. nächsten Wortes; Sonderzeichen wie Kommata zählen dabei nicht als eigene Wörter; nur im Kommandomodus   |
| <b>b</b> , <b>B</b>                                    | ( <i>backward-word</i> , <i>vi-word-back</i> ) auf Anfang des aktuellen bzw. vorherigen Wortes; nur im Kommandomodus                                      |
| <b>f</b>                                               | ( <i>vi-char-fwd</i> ) Cursor vorwärts auf Zeichen in aktueller Kommandozeile positionieren, das nachfolgend eingegeben wird; nur im Kommandomodus        |
| <b>F</b>                                               | ( <i>vi-char-back</i> ) Cursor rückwärts auf Zeichen in aktueller Kommandozeile positionieren, das nachfolgend eingegeben wird; nur im Kommandomodus      |
| <b>;</b>                                               | ( <i>vi-repeat-char-fwd</i> ) letztes <i>vi-char-fwd</i> bzw. <i>vi-char-back</i> in gleiche Richtung wiederholen; nur im Kommandomodus                   |
| <b>,</b>                                               | ( <i>vi-repeat-char-back</i> ) letztes <i>vi-char-fwd</i> bzw. <i>vi-char-back</i> in umgekehrter Richtung wiederholen; nur im Kommandomodus              |
| <b>t</b>                                               | ( <i>vi-charto-fwd</i> ) Cursor vorwärts vor Zeichen in aktueller Kommandozeile positionieren, das nachfolgend eingegeben wird; nur im Kommandomodus      |
| <b>T</b>                                               | ( <i>vi-charto-back</i> ) Cursor rückwärts hinter Zeichen in aktueller Kommandozeile positionieren, das nachfolgend eingegeben wird; nur im Kommandomodus |
| <b>nl</b>                                              | <i>n</i> Zeichen nach rechts (auch <i>n Pfeil-rechts</i> oder <i>n Leerzeichen</i> ); nur im Kommandomodus                                                |
| <b>nh</b>                                              | <i>n</i> Zeichen nach links (auch <i>n Pfeil-links</i> oder <i>n^H</i> ); nur im Kommandomodus                                                            |
| <b>nW</b>                                              | nach rechts um <i>n</i> Worte weiter; nur im Kommandomodus                                                                                                |
| <b>nB</b>                                              | nach links um <i>n</i> Worte; nur im Kommandomodus                                                                                                        |

| <i>Löschen</i>                                       |                                                                                                                                                                                                                                                                    |
|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Backspace</i> , <b>X</b>                          | ( <i>backward-delete-char</i> ) Zeichen links vom Cursor löschen; im Einfügemodus <b>^H</b> bzw. <i>Backspace</i>                                                                                                                                                  |
| <b>x</b>                                             | ( <i>delete-char</i> ) Zeichen an Cursorposition löschen; nur im Kommandomodus                                                                                                                                                                                     |
| <b>d</b>                                             | ( <i>vi-del-meta</i> ) leitet eine Löschaktion ein; z.B. löscht <b>dd</b> die ganze Zeile und <b>dw</b> das nächste Wort; nur im Kommandomodus                                                                                                                     |
| <b>D</b> , <b>^K</b>                                 | ( <i>kill-line</i> ) Ab Cursor bis Zeilenende löschen; <b>D</b> nur im Kommandomodus                                                                                                                                                                               |
| <b>^W</b>                                            | ( <i>backward-delete-word</i> ) Ab Cursor (nach links) bis Wortanfang löschen                                                                                                                                                                                      |
| <b>^U</b>                                            | ( <i>backward-kill-line</i> ) Ab Cursor bis zum Zeilenanfang löschen                                                                                                                                                                                               |
| <i>n Backspace</i>                                   | <i>n</i> Zeichen links vom Cursor löschen; nur im Kommandomodus                                                                                                                                                                                                    |
| <i>nX</i>                                            | <i>n</i> Zeichen links vom Cursor löschen; nur im Kommandomodus                                                                                                                                                                                                    |
| <i>nx</i>                                            | <i>n</i> Zeichen ab Cursor (nach rechts) löschen; nur im Kommandomodus                                                                                                                                                                                             |
| <i>ndw</i>                                           | ab Cursor (nach rechts) <i>n</i> Wörter löschen; nur im Kommandomodus                                                                                                                                                                                              |
| <i>n^W</i>                                           | ab Cursor (nach links) <i>n</i> Wörter löschen; nur im Kommandomodus                                                                                                                                                                                               |
| <i>Editieren von History-Kommandos</i>               |                                                                                                                                                                                                                                                                    |
| <b>-</b> , <b>k</b> , <b>^P</b> , <i>Pfeil-oben</i>  | ( <i>up-history</i> ) vorherige Kommandozeile aus History-Liste holen; <b>-</b> und <b>k</b> nur im Kommandomodus                                                                                                                                                  |
| <b>+</b> , <b>j</b> , <b>^N</b> , <i>Pfeil-unten</i> | ( <i>down-history</i> ) nächste Kommandozeile aus History-Liste holen; <b>+</b> und <b>j</b> nur im Kommandomodus                                                                                                                                                  |
| <b>K</b>                                             | ( <i>history-search-backward</i> ) Rückwärts in History-Liste nach Zeile suchen, die mit bisher eingegebenem String beginnt. Bei jedem neuen <b>K</b> wird sofort die nächste vorherige Zeile angezeigt, die mit dem gleichen String beginnt; nur im Kommandomodus |
| <b>J</b>                                             | ( <i>history-search-forward</i> ) wie <i>history-search-backward</i> , nur vorwärts in History-Liste suchen; nur im Kommandomodus                                                                                                                                  |
| <i>/text</i>                                         | ( <i>vi-search-back</i> ) nach <i>text</i> in History-Liste rückwärts suchen. Wird ein solcher <i>text</i> gefunden, wird die entsprechende Kommandozeile angezeigt; nur im Kommandomodus                                                                          |
| <b>N</b>                                             | ( <i>vi-repeat-search-back</i> ) letzte <i>/</i> -Suche wiederholen; nur im Kommandomodus                                                                                                                                                                          |
| <i>?text</i>                                         | ( <i>vi-search-fwd</i> ) nach <i>text</i> in History-Liste vorwärts suchen. Wird ein solcher <i>text</i> gefunden, wird die entsprechende Kommandozeile angezeigt; nur im Kommandomodus                                                                            |
| <b>n</b>                                             | ( <i>vi-repeat-search-fwd</i> ) letzte <i>?-</i> Suche wiederholen; nur im Kommandomodus                                                                                                                                                                           |

|                      |                                                                                   |
|----------------------|-----------------------------------------------------------------------------------|
| <i>n-Pfeil-oben</i>  | <i>n.te</i> vorherige Kommandozeile aus History-Liste holen; nur im Kommandomodus |
| <i>n-Pfeil-unten</i> | <i>n.te</i> nächste Kommandozeile aus History-Liste holen; nur im Kommandomodus   |

#### Automatische Vervollständigungen

|                        |                                                                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Tab</i> , <b>^I</b> | ( <i>vi-cmd-mode-complete</i> ) vervollständigt ein Wort ähnlich wie in Kapitel 8.3.2 beschrieben; Editor-Kommando für Einfügemodus ist <i>complete-word</i>         |
| <b>v</b> , <b>V</b>    | ( <i>expand-variables</i> ) vervollständigt den Variablennamen links vom Cursor; siehe auch Variable-Substitutionen; nur im Kommandomodus                            |
| <b>*</b> , <b>^G</b>   | ( <i>expand-glob</i> ) expandiert das Expandierungszeichen im Wort vor dem Cursor; im Einfügemodus nur <b>^G</b>                                                     |
| <b>!</b>               | ( <i>expand-history</i> ) History-Substitution für aktuelles Wort durchführen; siehe auch History-Substitution und Variable <b>autoexpand</b> ; nur im Kommandomodus |
| <b>^X</b>              | ( <i>expand-line</i> ) wie <i>expand-history</i> , nur daß History-Substitutionen für jedes Wort durchgeführt werden; nur im Kommandomodus                           |

#### Auflisten von Vervollständigungsmöglichkeiten

|           |                                                                                                                                              |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>^D</b> | ( <i>list-choices</i> ) listet (mit <i>ls -f</i> ) alle möglichen Vervollständigungen für das aktuelle Wort wie in Kapitel 8.3.2 beschrieben |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------|

#### Letzte Änderung rückgängig machen

|          |                                                                            |
|----------|----------------------------------------------------------------------------|
| <b>u</b> | ( <i>vi-undo</i> ) Letzte Änderung rückgängig machen; nur im Kommandomodus |
|----------|----------------------------------------------------------------------------|

#### Sonstiges

|                                       |                                                                                                                            |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>^T</b>                             | ( <i>transpose-chars</i> ) Zeichen an Cursorposition mit vorherigem vertauschen; nur im Einfügemodus                       |
| <b>~</b>                              | ( <i>change-case</i> ) Aktuelles Cursorzeichen von Groß- in Kleinbuchstaben umwandeln bzw. umgekehrt; nur im Kommandomodus |
| <b>r</b>                              | ( <i>vi-replace-char</i> ) Zeichen an Cursorposition durch das nächste eingegebene Zeichen ersetzen; nur im Kommandomodus  |
| <b>^L</b>                             | ( <i>clear-screen</i> ) Bildschirm löschen und aktuelle Zeile oben anzeigen                                                |
| <b>^R</b>                             | ( <i>redisplay</i> ) Bildschirm löschen und erneut anzeigen                                                                |
| <b>^J</b> , <b>^M</b> , <i>Return</i> | ( <i>newline</i> ) Kommandozeile ausführen                                                                                 |
| <b>^V</b>                             | ( <i>quoted-insert</i> ) Ausschalten des nachfolgenden Sonderzeichens; nur im Einfügemodus                                 |

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>M-?</b>              | ( <i>run-help</i> ) durchsucht die Dokumentation nach dem aktuellen Kommando. Diese Dokumentation sollte kurz sein, da keine seitenweise Ausgabe hierfür zur Verfügung steht. Zudem sollte sie sich in Dateien mit Namen wie <i>kommando.help</i> , <i>kommando.1</i> , <i>kommando.6</i> , <i>kommando.8</i> oder <i>kommando</i> befinden. Diese Dateien sollten sich in einem Directory befinden, das in der Environment-Variablen <b>HPATH</b> angegeben ist. Es wird grundsätzlich immer nur die erste gefundene Datei ausgegeben. |
| <b>a,b,c...,1,2....</b> | ( <i>self-insert-command</i> ) Alle diesem Kommando zugeteilten Zeichen werden bei Eingabe im Einfügemodus an der entsprechenden Stelle eingefügt                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>M-[ , M-O , O</b>    | ( <i>sequence-lead-in</i> ) zeigt an, daß das folgende Zeichen eine Multizeichen-Sequenz ist.                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>^@ , ...</b>         | ( <i>undefined-key</i> ) alle Tastenkombinationen, die keinem Editor-Kommando zugeteilt sind. Wird eine solche Tastenkombination eingegeben, ertönt der Piepston.                                                                                                                                                                                                                                                                                                                                                                       |

## 8.4 Lexikalische Struktur

### Aufteilen einer Kommandozeile in einzelne Wörter

Die tcsh teilt Eingabezeilen in einzelne Wörter auf. Als Worttrennzeichen werden dabei neben Leer- und Tabulatorzeichen noch die folgenden Sonderzeichen verwendet: **&**, **|**, **;**, **<**, **>**, **(**, **)**, **&&**, **| |**, **<<** und **>>**.

### Kommentare

Wenn die Standardeingabe der tcsh nicht auf ein Terminal eingestellt ist, kann mit **#** ein Kommentar eingeleitet werden. Der Rest der Zeile nach **#** wird dann von der tcsh ignoriert.

### Quoting

Die Bedeutung von Sonderzeichen wie z.B. *Leerzeichen*, *Tabzeichen* oder **&** kann durch Voranstellen eines Backslashes (**\**) ausgeschaltet werden. Um in einem ganzen String die Sonderbedeutung von solchen speziellen Zeichen auszuschalten, muß dieser String mit **".."** bzw. **'..'** geklammert werden. Ist vor einem Neuezeile-Zeichen ein **\** angegeben, wird diese Angabe von der tcsh durch ein Leerzeichen ersetzt. Innerhalb von **'..'** oder **".."** wird eine solche Angabe durch ein Neuezeile-Zeichen ersetzt.

Bis auf History-Substitutionen können alle Substitutionen dadurch ausgeschaltet werden, daß man das entsprechende Wort oder eventuell nur das betreffende Sonderzeichen mit **\** oder **'..'** ausschaltet. History-Substitutionen dagegen können nur unter Verwendung des Backslashes (**\**) ausgeschaltet werden.

Innerhalb von `".."` werden bis auf Variable-Substitution (`$`) und Kommando-Substitution (``..``) alle anderen Substitutionen ausgeschaltet.

### Die Variable `backslash_quote`

Ist die Variable **`backslash_quote`** gesetzt, kann mit Backslash (`\`) die Bedeutung der Sonderzeichen `\`, `".."` und `'..'` ausgeschaltet werden.

## 8.5 Substitutionen

Nachfolgend werden die Substitutionen in der Reihenfolge beschrieben, wie sie von der `tcsh` durchgeführt werden.

### 8.5.1 History-Substitutionen

Jede eingegebene Kommandozeile wird von der `tcsh` in der History-Liste aufgehoben.

Mit dem builtin-Kommando **`history`** ist es möglich, sich die ganze History-Liste anzeigen zu lassen, sie eventuell in der Datei speichern zu lassen, oder sie vollständig zu löschen.

Beim Abmelden speichert die `tcsh` die History-Liste in der Datei `~/.history`, die beim nächsten Anmelden dann wieder gelesen wird, so daß die in dieser Sitzung eingegebenen Kommandos dann wieder verfügbar sind.

Mit dem Setzen der Variablen **`history=n`** kann festgelegt werden, wie viele Kommandozeilen in der History-Liste aufzuheben sind. Mit dem Setzen der Variable **`savehist=n`** kann festgelegt werden, wie viele Kommandozeilen in der Datei `~/.history` zu speichern sind. So legt z.B.

**`set history = 40 savehist = 30`**

fest, daß immer die letzten 40 Kommandozeilen in der History-Liste zu halten sind, aber nur 30 Kommandozeilen in der Datei `~/.history` zu speichern sind. Der Wert von **`savehist`** darf niemals größer als der von **`history`** sein.

Ist die Variable **`histdup`** gesetzt, werden aufeinanderfolgende Kommandozeilen, die völlig identisch sind, nur einmal in der History-Liste aufgehoben.

Soll auf eine zuvor eingegebene Kommandozeile wieder zugegriffen werden, muß das Ausrufezeichen `!` angegeben werden.

Dabei kann nun folgendes angegeben werden:

| Angabe             | Aus History-Liste wird in aktueller Kommandozeile eingesetzt                                                                                                                   |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>!n</b>          | Kommandozeile mit der event number <i>n</i> .                                                                                                                                  |
| <b>!-n</b>         | <i>n</i> .te vorherige Kommandozeile.                                                                                                                                          |
| <b>!!</b>          | unmittelbar zuvor eingegebene Kommandozeile; !! entspricht also <b>!-1</b> .                                                                                                   |
| <b>!string</b>     | Letzte Kommandozeile, die mit <i>string</i> beginnt.                                                                                                                           |
| <b>!?string[?]</b> | Letzte Kommandozeile, die <i>string</i> enthält.                                                                                                                               |
| <b>!#</b>          | bisher eingegebene aktuelle Zeile. Mit dieser Aufrufform ist es möglich, auf zuvor (weiter links) eingegebene Wörter wieder zuzugreifen, ohne diese erneut eingeben zu müssen. |

**!** verliert seine Sonderbedeutung, wenn vor ihm ein **\** steht oder ihm ein Leer-, Tab-, Neuezeile-Zeichen, = oder **(** folgt.

Die spezielle Angabe **^alt^neu^** ersetzt in der vorherigen Kommandozeile den durch *alt* abgedeckten Text durch den Text *neu*. Diese Angabe ist somit äquivalent zu **!!:s/alt/neu/**.

Eine Angabe wie

**!ltx**

würde in der History-Liste nach einer Zeile suchen, die mit *ltx* beginnt, während die Angabe

**!{l}tx**

in der History-Liste nach einer Zeile suchen, die mit *l* beginnt und *tx* endet, wie z.B. **ls \*.tx**.

Während die C-Shell z.B. bei der Angabe

**!5x**

die 5. Zeile in der History-Liste holt und ihr ein *x* anhängt, sucht die tcsh in diesem Fall in der History-Liste nach einer Zeile, die mit *5x* beginnt. Um in diesem Fall das Verhalten der C-Shell nachzubilden, muß in der tcsh

**!\5x**

eingegeben werden.

Es ist auch möglich, daß man nicht die ganze Kommandozeile, sondern nur Teile davon einsetzen läßt:

**!angabe[:auswahl][:modifikator(en)]**

Für *langabe* muß eine der sechs zuvor beschriebenen Formen verwendet und für *auswahl* kann folgendes angegeben werden:

| Auswahl    | Wortauswahl in entsprechender Kommandozeile                                                              |
|------------|----------------------------------------------------------------------------------------------------------|
| <i>n</i>   | <i>n</i> .tes Wort; <b>0</b> ist das erste Wort, <b>1</b> das zweite Wort, <b>2</b> das dritte Wort usw. |
| <b>^</b>   | erstes Argument (nicht Kommandoname)                                                                     |
| <b>\$</b>  | letztes Argument                                                                                         |
| <b>%</b>   | das durch eine zuvor angegebene Konstruktion <i>?string?</i> ausgewählte Wort                            |
| <i>m-n</i> | <i>m</i> bis <i>n</i> .tes Wort; Wortzählung beginnt wieder bei 0                                        |
| <i>-n</i>  | entspricht <b>0-n</b>                                                                                    |
| <i>m-</i>  | <i>m</i> bis vorletztes Wort                                                                             |
| <b>*</b>   | alle Wörter außer 0.Wort; entspricht <b>^-\$</b>                                                         |
| <i>n*</i>  | entspricht <i>n-\$</i>                                                                                   |

Der Doppelpunkt : vor *auswahl* kann weggelassen werden, wenn *auswahl* mit **^**, **\$**, **\***, **-** oder **%** beginnt.

Die über die Angabe von *auswahl* ausgewählten Wörter könnten durch Angabe von *modifikatoren* noch modifiziert werden:

| Modifikator       | Auswirkung                                                                                                                                                                                                                                                                                                    |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>h</b>          | ( <b>head</b> ) liefert den Pfadnamen des Wortes.                                                                                                                                                                                                                                                             |
| <b>t</b>          | ( <b>tail</b> ) liefert den Basisnamen des Wortes (Gegenstück zu <b>h</b> ).                                                                                                                                                                                                                                  |
| <b>r</b>          | ( <b>remove</b> ) schneidet vom Wort eine vorhandene Endung .xxx ab.                                                                                                                                                                                                                                          |
| <b>e</b>          | ( <b>extension</b> ) liefert vom Wort nur eine eventuell vorhandene Endung .xxx.                                                                                                                                                                                                                              |
| <b>u</b>          | ( <b>uppercase</b> ) den ersten Kleinbuchstaben in Großbuchstaben umwandeln.                                                                                                                                                                                                                                  |
| <b>l</b>          | ( <b>lowercase</b> ) den ersten Großbuchstaben in Kleinbuchstaben umwandeln.                                                                                                                                                                                                                                  |
| <i>s/alt/neu/</i> | ( <b>substitute</b> ) ersetzt den durch <i>alt</i> abgedeckten Text durch <i>neu</i> . Für <i>alt</i> dürfen die bei <b>ed</b> zugelassenen regulären Ausdrücke verwendet werden.<br><br>Wird das Zeichen <b>&amp;</b> in <i>neu</i> angegeben, wird hierfür der durch <i>alt</i> abgedeckte Text eingesetzt. |



| Modifikator  | Auswirkung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | <p>Wird für <i>neu</i> keinerlei Text angegeben, wird dafür das <i>neu</i> aus einem vorherigen <i>s/alt/neu/</i> oder aus einer vorherigen Angabe <i>!string?</i> genommen.</p> <p>Statt / darf jedes beliebige Trennzeichen verwendet werden, solange es nicht in <i>alt</i> oder <i>neu</i> vorkommt.</p> <p>Wenn nach <i>s/alt/neu/</i> sofort ein Neuezeile-Zeichen angegeben wird, darf das letzte / auch weggelassen werden.</p> <p>Um die Bedeutung von Sonderzeichen wie <b>&amp;</b> oder / in <i>alt</i> oder <i>neu</i> auszuschalten, muß diesen ein \ vorangestellt werden.</p> |
| <b>&amp;</b> | wiederholt die letzte mit <i>s/alt/neu/</i> angegebene Substitution.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>g</b>     | ( <i>global</i> ) Modifikator wird nicht nur auf das erste Wort, sondern auf alle ausgewählten Wörter angewendet. <b>g</b> ist dabei vor dem Modifikator ( <b>s</b> , <b>&amp;</b> ) anzugeben.                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>a</b>     | den nachfolgenden Modifikator so oft wie möglich auf ein einzelnes Wort anwenden. <b>a</b> und <b>g</b> können zusammen angegeben werden, um einen Modifikator auf ganze Zeilen anzuwenden. Momentan kann <b>a</b> und <b>s</b> zusammen zu einer Endlosschleife führen wie z.B. in <b>:as/x/xx/</b> .                                                                                                                                                                                                                                                                                        |
| <b>p</b>     | ( <i>print</i> ) die neu entstandene Kommandozeile wird nur angezeigt, aber nicht ausgeführt.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>q</b>     | ( <i>quote</i> ) für die Kommandozeile wird Quoting vorgenommen, so daß keine weiteren Substitutionen oder Dateinamenexpansionen stattfinden.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>x</b>     | wie <b>q</b> , nur daß der entsprechende Wert in mehrere Worte aufgeteilt wird. Trennzeichen sind dabei Leer-, Tabulator- und Neuezeile-Zeichen.                                                                                                                                                                                                                                                                                                                                                                                                                                              |

Nur wenn **g** als erster Modifikator angegeben ist, werden die Modifikationen für alle Wörter durchgeführt, ansonsten werden sie nur für das erste Wort vorgenommen.

Der Modifikator **a** ist neu in der tcsh. Er bewirkt, daß die folgenden Modifikatoren mehrmals auf dasselbe Wort angewendet werden, wie folgende Beispiele verdeutlichen:

```
> set liste = (a:b:c d:e:f)
> echo $liste:s/:-/
a-b:c d:e:f
> echo $liste:gs/:-/
a-b:c d-e:f
> echo $liste:as/:-/
a-b-c d:e:f
> echo $liste:ags/:-/
a-b-c d-e-f
```

An diesen Beispielen wird auch eine weitere Neuheit der tcsh deutlich – das Anwenden der Modifikators **s** auf Variablen.

Ein weiteres Beispiel soll die Wirkungsweise des Modifikators **a** verdeutlichen:

```
> set ort = erlangen.franken.deutschland.europa
> echo $ort
erlangen.franken.deutschland.europa
> echo $ort:r
erlangen.franken.deutschland
> echo $ort:ar
erlangen
```

Während in der C-Shell nur ein Modifikator für jede History- oder Variablen-Expandierung angewendet werden kann, ist es in der tcsh möglich, mehr als einen Modifikator anzuwenden wie z.B.

```
> cp *.c ~/src/xxx.c
> cc -o !$:t:r !$
cc -o xxx ~/src/xxx.c
```

In der C-Shell würde obige Eingabe resultieren in:

```
cc -o xxx ~/src/xxx.c:r
```

Eine Substitution, der ein Doppelpunkt folgt, muß in **{.}** angegeben werden wie z.B.

```
> cp xxx ~hh/bin/zaehle
> setenv PATH !$:h:$PATH
Bad ! modifier: $.
> setenv PATH ![-2:h]:$PATH
setenv PATH ~hh/bin:$PATH
```

Während der erste **setenv**-Aufruf in der C-Shell erfolgreich wäre, schlägt er in der tcsh fehl, da die tcsh nach dem zweiten Doppelpunkt einen weiteren Modifikator erwartet.

Wird in der Shellvariablen **prompt** das Zeichen **!** angegeben, wird bei der Ausgabe des Prompts hierfür immer die aktuelle Kommandozeilennummer (*event number*) eingesetzt.

Die tcsh hebt die eingegebenen Kommandozeilen immer in zwei verschiedenen Darstellungsformen in der History-Liste auf: mit und ohne expandierte History-Substitutionen. Um zwischen beiden Darstellungsformen hin- und herschalten zu können, steht das Editor-Kommando *toggle-literal-history* (voreingestellt auf **M-R**) zur Verfügung. Ist die Shellvariable **histlit** gesetzt, ist die nicht expandierte Form eingestellt.

```
> echo !:s/hans/fritz; ls
Modifier failed.
> ^P
> echo !:s/hans/fritz; ls
> ^N
> unset histlit
> echo !:s/hans/fritz; ls
Modifier failed.
> ^P
> echo unset hitlist [M-R]
> echo !:s/hans/fritz; ls
```

Die tcsh unterstützt auch die History-Konstruktion **!#**. Diese nicht dokumentierte History-Konstruktion enthält immer die Wörter der aktuellen Kommandozeile:

```
> touch datei
> mv datei alt!#:1
mv datei altdatei
> echo --- !#:1!#:1!#:1
echo --- -----

```

Während in der C-Shell keine Überprüfung bei Rekursion stattfindet, wird in der tcsh nach 10 Rekursionen abgebrochen.

## 8.5.2 Alias-Substitutionen

Die tcsh unterhält eine Liste von Aliase, die mit den Kommandos **alias** und **unalias** gesetzt, aufgelistet oder gelöscht werden können. Nachdem eine Kommandozeile in einfache Kommandos aufgeteilt wurde, wird von links nach rechts für jedes Kommando das erste Wort geprüft, ob es sich dabei um ein Alias handelt. Wenn ja, wird hierfür der entsprechende Alias-Text eingesetzt. Enthält der Alias-Text eine History-Konstruktion, wird hierfür eine History-Substitution durchgeführt, und zwar so, als ob das originale Kommando die vorherige Eingabezeile wäre.

```
> tcsh -x [-x bewirkt das Anzeigen von Kommandozeilen vor Ausführung]
> alias ls 'ls -l '
alias ls ls -l
> ls /etc
ls -l /etc
.....
.....
> alias nameinfo 'grep \!^ /etc/passwd'
alias nameinfo 'grep \!^ /etc/passwd'
> nameinfo hh
grep hh /etc/passwd
hh:b9U/9RU.X29JQ:2021:2:Helmut Herold:/home/hh:/bin/bash
```

Eine Alias-Substitution wird solange wiederholt, bis das erste Wort des Kommandos kein Alias-Name mehr ist. Um eine Endlos-Rekursion zu unterbinden, wird aber nur dann eine erneute Alias-Substitution für das erste Wort durchgeführt, wenn es sich nicht geändert hat; siehe auch Alias **ls** im vorherigen Beispiel. Eventuell versteckte Rekursionen werden von der `tcsh` erkannt und durch eine Fehlermeldung angezeigt.

Es gibt auch eine Reihe von speziellen, von der `tcsh` vordefinierten Aliase (siehe dazu Kapitel 8.12).

### 8.5.3 Variablen-Substitutionen

Die `tcsh` unterhält eine Liste von Variablen, deren Werte entweder kein, ein oder mehrere Wörter sind. Mit den Kommandos **set** und **unset** können Variablen neue Werte zugewiesen, ihre aktuellen Inhalte aufgelistet oder ihre momentanen Werte gelöscht werden.

Daneben unterhält die `tcsh` noch eine eigene Liste von Environment-Variablen, die mit den Kommandos **printenv**, **setenv** und **unsetenv** entweder aufgelistet oder verändert werden können.

Mit **set -r** kann in der `tcsh` eine Variable als *nur-lesbar* (*readonly*) festgelegt werden. Nachdem eine Variable einmal auf *nur-lesbar* gesetzt wurde, kann sie nicht wieder später auf *beschreibbar* gesetzt werden. Environment-Variablen können niemals auf *nur-lesbar* gesetzt werden.

```
> set -r zahlen = (1 2 3)
> unset zahlen
unset: $zahlen is read-only.
> set zahlen = 7
unset: $zahlen is read-only.
> set -r zahlen = 8
unset: $zahlen is read-only.
> echo $zahlen
1 2 3
```

Wird **set -r** ohne Argumente aufgerufen, werden alle *readonly*-Variablen ausgegeben.

Die einfachste Zugriffsmöglichkeit auf den Wert einer Variablen ist

**\$variable**

Die `tcsh` ersetzt diesen Ausdruck dabei durch alle Worte des Wertes von *variable* (kann auch eine mit **setenv** definierte Environment-Variable sein); die einzelnen Worte werden dabei durch Leerzeichen getrennt.

Weitere Zugriffsmöglichkeiten auf den Wert einer Variablen sind nachfolgend angegeben:

|                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><code>\${variable}</code></b> <sup>h</sup>                                                                                                              | entspricht <code>\$variable</code> .<br><br>Diese Angabe muß immer dann verwendet werden, wenn entweder dem Variablennamen ein Buchstabe, eine Ziffer oder ein Unterstrich folgt, oder auf Positionsparameter mit mehr als einer Ziffer (größer als 9) zuzugreifen ist.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b><code>\$variable</code></b><br><b><code>[index]</code></b> <sup>h</sup><br><b><code>\${variable</code></b><br><b><code>[index]}</code></b> <sup>h</sup> | Zugriff auf ein oder mehrere Worte.<br><br>Der kleinstmögliche <code>index</code> ist immer 1.<br><br>Für <code>index</code> kann angegeben werden:<br><br>eine Zahl <code>n</code> <code>n</code> .te Wort<br><br><code>m-</code> Worte <code>m</code> bis <code>n</code><br><br><code>-n</code> entspricht <code>1-n</code><br><br><code>m-</code> entspricht <code>m-\${#variable}</code><br><br><code>*</code> alle Worte<br><br>Die Angabe eines leeren Bereiches resultiert nur dann nicht in einem Fehler, wenn der zweite Index weggelassen wird oder aber im erlaubten Bereich liegt.                                                                                                      |
| <b><code>\$0</code></b> <sup>h</sup>                                                                                                                       | Skriptname                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b><code>\$n</code></b> <sup>h</sup><br><b><code>\${n}</code></b> <sup>h</sup>                                                                             | entspricht <code>\$argv[n]</code> bzw. <code>\${argv[n]}</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b><code>\$argv</code></b> <sup>h</sup>                                                                                                                    | Variable, die die Positionsparameter enthält:<br><br><b><code>\$argv[0]</code></b> (nicht definiert; <b><code>\$0</code></b> dagegen enthält Skript-Namen)<br><br><b><code>\$argv[1]</code></b> entspricht <b><code>\$1</code></b><br><br>:<br><br><b><code>\$argv[9]</code></b> entspricht <b><code>\$9</code></b><br><br><b><code>\$argv[10]</code></b> entspricht <b><code>\${10}</code></b><br><br><b><code>\$argv[11]</code></b> entspricht <b><code>\${11}</code></b><br><br>:<br><br><b><code>\$argv[*]</code></b> entspricht <b><code>\$*</code></b><br><br><b><code>\$argv</code></b> entspricht <b><code>\$*</code></b><br><br><b><code>\$#argv</code></b> Anzahl der Positionsparameter. |
| <b><code>\$*</code></b> <sup>h</sup>                                                                                                                       | entspricht <b><code>\$argv</code></b> , was äquivalent zu <b><code>\$argv[*]</code></b> ist                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b><code>\$#variable</code></b><br><b><code>\${#variable}</code></b>                                                                                       | Anzahl der Worte in <code>variable</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

|                                          |                                                                                                                                                                                                                                                                      |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>\$#</b>                               | äquivalent zu <b>\$#argv</b> .                                                                                                                                                                                                                                       |
| <b>%variable</b><br><b>\${%variable}</b> | Anzahl der Zeichen in <i>variable</i> .                                                                                                                                                                                                                              |
| <b>%n</b><br><b>\${%n}</b>               | Anzahl der Zeichen in <b>\$argv[n]</b> .                                                                                                                                                                                                                             |
| <b>{?variable}</b><br><b>?variable</b>   | liefert 1, wenn <i>variable</i> gesetzt ist, und 0, wenn nicht.                                                                                                                                                                                                      |
| <b>\$?</b>                               | liefert 1, wenn <b>\$0</b> gesetzt ist, und 0, wenn nicht; in einer interaktiven Shell immer 0.                                                                                                                                                                      |
| <b>&lt;</b>                              | Hierfür wird eine von der Standardeingabe gelesene Zeile eingesetzt. Mit dieser Konstruktion ist es also möglich, eine Zeile von der Standardeingabe zu lesen.<br><br>Während die C-Shell für <b>&lt;</b> immer <b>&lt;:q</b> annimmt, gilt dies nicht für die tcsh. |
| <b>\$</b>                                | liefert die Prozeßnummer (PID) der aktuellen Shell.                                                                                                                                                                                                                  |
| <b>\$status</b><br><b>\$?</b>            | Exit-Status des letzten Vordergrund-Kommandos.                                                                                                                                                                                                                       |
| <b>\$child</b><br><b>!</b>               | Prozeßnummer (PID) des letzten Hintergrund-Kommandos.                                                                                                                                                                                                                |

Außer **:p** dürfen alle unter *History-Substitutionen* (Kapitel 8.5.1) beschriebenen Modifikatoren nur bei den mit <sup>h</sup> gekennzeichneten Konstruktionen verwendet werden.

Mit dem Editor-Kommando *expand-variables* (voreingestellt auf **^X-\$**) können Variablen bereits bei der Eingabe in der Kommandozeile expandiert werden.

### 8.5.4 Kommando-, Dateinamen- und Directorystack-Substitutionen

Diese Substitutionen werden getrennt für jedes einzelne Wort in der Kommandozeile durchgeführt.

#### Kommando-Substitution

Eine Kommando-Substitution ist wie in der csh mit **`...`** anzugeben. Die Ausgabe des in **`...`** angegebenen Kommandos wird in einzelne Wörter aufgeteilt, wobei Leer-, Tab- und Neuezeile-Zeichen entfernt werden, bevor dann diese einzelnen Wörter an der entsprechenden Stelle in der Kommandozeile (mit Leerzeichen getrennt) eingesetzt werden. Für diese Ausgabe findet nachfolgend dann aber noch Variable- und Kommando-Substitution statt.

Befindet sich eine Kommando-Substitution ``...`` innerhalb von `"..."`, bleiben die dort angegebenen Leer- und Tabzeichen erhalten, nur eventuell angegebene Neuezeile-Zeichen werden entfernt.

### Dateinamen-Substitution (Dateinamenexpansion)

Enthält ein Wort eines der Zeichen `*`, `?`, `[`, `{` oder beginnt es mit `~` (Tilde), wird für dieses Wort Dateinamen-Substitution (*globbing*) durchgeführt. Wird ein solches Wort gefunden, betrachtet die tcsh dieses als ein sogenanntes *pattern*, das eine Vielzahl von Dateinamen abdecken kann.

Jedes in der Kommandozeile gefundene Pattern wird dann von der tcsh expandiert, d.h. durch alle Dateinamen (alphabetisch sortiert) ersetzt, die es abdeckt.

|                                                      |                                                                                                                                                                                                                                                                                          |
|------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>*</code>                                       | steht für »eine beliebige Zeichenfolge« (auch die leere)                                                                                                                                                                                                                                 |
| <code>?</code>                                       | steht für »ein beliebiges einzelnes Zeichen«                                                                                                                                                                                                                                             |
| <code>[...]</code>                                   | steht für »eines der in [...] angegebenen Zeichen«.<br>Bei der Angabe von Zeichen innerhalb von [...] sind auch Bereichsangaben wie <code>[A-Z]</code> oder <code>[0-9]</code> erlaubt.                                                                                                  |
| <code>[^...]</code>                                  | steht für »eines der nicht in [^...] angegebenen Zeichen«<br>Bei der Angabe von Zeichen innerhalb von [^...] sind auch Bereichsangaben wie <code>[^A-Z]</code> oder <code>[^0-9]</code> erlaubt.                                                                                         |
| <code>{wort<sub>1</sub>,wort<sub>2</sub>,...}</code> | Jedes der in {...} mit Komma getrennten Worte wird eingesetzt; z.B. würde <code>prog.{c,o,txt}</code> die Dateien <code>prog.c</code> , <code>prog.o</code> und <code>prog.txt</code> und <code>{add1,subst}.c</code> die Dateien <code>add1.c</code> und <code>subst.c</code> abdecken. |
| <code>~</code>                                       | (Tilde) Hierfür setzt die tcsh den Pfadnamen des Home-Directorys ein, wenn <code>~</code> als einziges Zeichen eines Wortes vorkommt oder am Anfang eines Wortes steht und ihm / folgt.                                                                                                  |
| <code>~loginname</code>                              | (Tilde) Hierfür setzt die tcsh den Pfadnamen des Home-Directorys des Benutzers <i>loginname</i> ein, wenn diese Konstruktion am Anfang eines Wortes steht.                                                                                                                               |

Einige Zeichenfolgen in Dateinamen werden nur dann abgedeckt, wenn sie explizit im entsprechenden Pattern angegeben wurden:

`.` (Punkt) am Anfang eines Dateinamens  
`/.`  
`/`

Neu in der tcsh ist, daß ganze pattern mit ^ negiert werden können:

```
> echo *
acker wald waldblume waldmeister walker wiese
> echo ^wald*
acker walker wiese
> echo ^w*
acker
```

Wird ^ auf Wörter angewendet, die kein \*, ?, [, { oder ~ enthalten, wird die Negation nicht richtig durchgeführt.

Ist die Variable **nonomatch** nicht gesetzt, wertet die tcsh ein Metazeichen für Dateinamenexpansion (\* ? [..] [^..] {..}), das keinen Dateinamen abdeckt, als Fehler und meldet dies.

Ist die Variable **noglob** gesetzt, wird die Dateinamenexpansion ausgeschaltet.

Mit dem Editor-Kommando *expand-glob* (voreingestellt auf ^X-\*) kann für ein einzelnes Wort bereits während dem Editieren der Kommandozeile Dateinamenexpansion durchgeführt werden.

### Directorystack-Substitution

Der Directorystack ist eine Liste von Directorynamen, die ab 0 durchnummeriert sind. Mit dem builtin-Kommando **dirs** kann der Inhalt des Directorystacks ausgegeben, verändert oder gelöscht werden. Mit dem builtin-Kommando **popd** kann zu einem Directory des Directorystacks gewechselt bzw. ein Directory aus dem Directorystack entfernt werden. Mit dem builtin-Kommando **pushd** kann ein neues Directory zum Directorystack hinzugefügt bzw. der Inhalt des Directorystacks rotiert oder einzelne Einträge vertauscht werden.

Die beiden Variablen **savedirs** und **dirsfile** können entsprechend gesetzt werden, um einen Directorystack in eine Datei zu schreiben, so daß er beim nächsten Login wieder zur Verfügung gestellt werden kann.

Die Variable **dirstack** enthält immer die einzelnen Einträge des Directorystacks (**\$dirstack[1]**, **\$dirstack[2]**, ...), die man sich mittels **echo** ausgeben lassen kann. Ein Verändern der einzelnen Worte hat dabei auch die entsprechende Auswirkung auf den Directorystack.

Das Zeichen = gefolgt von einer Nummer wird immer zum entsprechenden Eintrag des Directorystacks expandiert, wobei -= sich immer auf den letzten Eintrag des Directorystacks bezieht. z.B.

```
> pwd
/home/hh
> pushd /etc
/etc ~
> pwd
```



```
/etc
> pushd /var/adm
/var/adm /etc ~
> pushd /usr/bin
/usr/bin /var/adm /etc ~
> pwd
/usr/bin
> dirs -v
0 /usr/bin
1 /var/adm
2 /etc
3 ~
> echo =1
/var/adm
> echo =0/wc
/usr/bin/wc
> echo =-
/home/hh
```

Die Variablen **nonomatch**, **noglob** und das Editor-Kommando *expand-glob* (voreingestellt auf **^X-\***) können ebenso auf den Directorystack wie bei Dateinamenexpandierung angewendet werden.

## 8.6 Ausführung von Kommandos

Die nächsten drei Unterkapitel beschreiben, wie die tcsh Kommandos ausführt und behandeln die Ein- und Ausgabe.

### 8.6.1 Einfache Kommandos, Pipelines und Listen

#### Einfache Kommandos

Ein *einfaches Kommando* ist eine Folge von Wörtern, die durch Leer- und/oder Tabulatorzeichen voneinander getrennt sind. Das erste Wort gibt den Namen des auszuführenden Kommandos an.

#### Pipelines

Eine *Pipeline* ist eine Folge von einem oder mehreren einfachen Kommandos, die mit **|** oder **|&** voneinander getrennt sind. Der exit-Status einer Pipeline ist der exit-Status des zuletzt in der Pipeline angegebenen Kommandos.

#### Kommando-Listen

Eine *Kommando-Liste* ist eine Folge von ein oder mehreren Pipelines, die durch die Metazeichen **;** **&** **&&** **||** voneinander getrennt sind. **&** und **;** dürfen dabei auch am Ende einer solchen Liste angegeben sein.

Die Vorrangregeln der hier angegebenen Trennzeichen von Pipelines in einer Liste sind: ( ; gleich & ) < ( && gleich | | )

Um eine Liste als einfaches Kommando behandeln zu können, muß diese mit ( ) geklammert werden.

### 8.6.2 Ausführung von builtin- und anderen Kommandos

Builtin-Kommandos werden in der aktuellen Shell ausgeführt. Ist in einer Pipeline ein Kommando (außer dem letzten) ein builtin-Kommando, wird die Pipeline in einer Subshell ausgeführt.

Geklammerte Kommandos werden immer in einer Subshell ausgeführt, wie z.B.

```
> pwd
/home/hh
> (cd /usr/lib; pwd); pwd
/usr/lib
/home/hh
> pwd
/home/hh
```

[nach Ausführung der Subshell befindet man sich wieder im alten Directory]

Dagegen würde man mit dem folgenden Kommando das Working-Directory wirklich wechseln

```
> cd /usr/lib; pwd
/usr/lib
> pwd
/usr/lib
```

[/usr/lib ist nun das neue Working-Directory]

Ist ein Kommando kein builtin-Kommando, startet die tcsh für die Ausführung des entsprechenden Kommandos eine Subshell. Dabei durchsucht die tcsh jedes in der Variable **path** angegebene Directory nach dem angegebenen Kommandonamen. Ist weder die Option **-t** noch die Option **-c** gesetzt, werden alle Kommandos der **path**-Directories in die interne Hashing-Tabelle eingetragen, so daß beim nächsten Kommandoaufruf nicht alle diese Directories erneut durchsucht werden müssen.

In den folgenden Fälle fügt die tcsh nur den Kommandonamen an den Pfadnamen des Working-Directorys und versucht dann diesen neu entstandenen Pfadnamen zur Ausführung zu bringen:

- ▶ Hashing-Mechanismus ist mit dem builtin-Kommando **unhash** ausgeschaltet
- ▶ Optionen **-c** oder **-t** sind gesetzt
- ▶ **path**-Directories, die nicht mit / beginnt

Hat die entsprechende Datei Ausführrechte, ist sie aber weder ein ausführbares Programm noch ein Shellskript mit `#!/interpreter` am Anfang, dann nimmt die `tcsh` an, daß es sich um ein Shellskript handelt und startet eine neue Subshell, um die Datei als Shellskript ausführen zu lassen. Soll ein anderes Programm als die `tcsh` ein solches Shellskript ausführen, kann dies mit dem builtin-Kommando **shell** festgelegt werden.

### 8.6.3 Ein- und Ausgabe

Die Standardein- und ausgabe eines Kommandos kann wie folgt umgelenkt werden:

- `>datei`      Standardausgabe in *datei* umlenken.
- `>>datei`     Standardausgabe am Ende von *datei* anfügen.
- `<datei`      Standardeingabe auf *datei* umlenken.
- `<<wort`     Hier-Dokument (*here document*): Standardeingabe besteht aus den nächsten Eingabezeilen bis zur ersten Zeile, die nur *wort* enthält.
- `>! datei`    wie `>`, nur daß eine bereits existierende *datei* auch überschrieben wird, wenn **noclobber** gesetzt ist.
- `>& datei`    Standardausgabe und Standardfehlerausgabe in *datei* umlenken.
- `>&! datei`   wie `>&`, nur daß eine bereits existierende *datei* auch dann überschrieben wird, wenn **noclobber** gesetzt ist.
- `>>! datei`   wie `>>`, nur daß auch in *datei* geschrieben wird, wenn *datei* noch nicht existiert und **noclobber** gesetzt ist.
- `>>& datei`   Standardausgabe und Standardfehlerausgabe an das Ende von Datei *datei* umlenken.
- `>>&! datei`   wie `>>&`, nur daß auch in *datei* geschrieben wird, wenn *datei* noch nicht existiert und **noclobber** gesetzt ist.

Für die *datei* wird Variablen-, Kommando- und Dateinamen-Substitution durchgeführt.

Um die Standardausgabe und Standardfehlerausgabe in unterschiedliche Dateien umzulenken, empfiehlt sich die folgende Konstruktion:

`(kommando_zeile > stdout_datei) >& stderr_datei`

Wird für *stdout\_datei* oder *stderr\_datei* der Name `/dev/tty` angegeben, erfolgt die entsprechende Ausgabe auf das Terminal.

## 8.7 tcsh-Ausdrücke

Ausdrücke werden zum einen bei den builtin-Kommandos **if**, **while** und **exit** und zum anderen bei arithmetischen Berechnungen mit dem builtin-Kommando **@** benötigt, wobei jedoch **@** eine eigene Syntax hat.

Wie in der Programmiersprache C wird der Wert eines arithmetischen Ausdrucks als TRUE gewertet, wenn er verschieden von 0, und als FALSE, wenn er 0 ist.

Für Ausdrücke können folgende C-ähnliche Operatoren verwendet werden:

|                           |                                                                                      |
|---------------------------|--------------------------------------------------------------------------------------|
| <b>-, ~</b>               | Minuszeichen, Bitweise Negation (Einerkomplement)                                    |
| <b>*, /, %</b>            | Multiplikation, Division, Modulo                                                     |
| <b>+, -</b>               | Addition, Subtraktion                                                                |
| <b>&lt;&lt;, &gt;&gt;</b> | Links-Shift, Rechts-Shift                                                            |
| <b>&lt;, &gt;</b>         | kleiner, größer                                                                      |
| <b>&lt;=, &gt;=</b>       | kleiner gleich, größer gleich                                                        |
| <b>==, !=</b>             | Vergleich von Strings auf Gleichheit / Ungleichheit                                  |
| <b>=~</b>                 | Vergleich von Strings auf Gleichheit (als rechter Operand ist ein pattern erlaubt)   |
| <b>!~</b>                 | Vergleich von Strings auf Ungleichheit (als rechter Operand ist ein pattern erlaubt) |
| <b>&amp;, ^,  </b>        | bitweises AND, XOR, OR                                                               |

Die Operatoren **==**, **!=**, **=~** und **!~** führen einen Stringvergleich durch. Die Operatoren **=~** und **!~** entsprechen weitgehend den Operatoren **==** und **!=**, nur daß sie als rechten Operanden auch ein pattern zulassen; dabei prüft die tcsh dann, ob dieses pattern den linken Operanden abdeckt (**=~**) oder nicht abdeckt (**!~**).

Zusätzlich können noch die folgenden Ausdrücke angegeben werden:

| Ausdruck               | liefert TRUE, wenn <i>datei</i> bzw. <i>n</i>                   |
|------------------------|-----------------------------------------------------------------|
| <b>-d <i>datei</i></b> | ein Directory ( <b>d</b> irectory) ist.                         |
| <b>-e <i>datei</i></b> | existiert ( <b>e</b> xistence).                                 |
| <b>-f <i>datei</i></b> | eine normale Datei ( <b>f</b> ile) ist.                         |
| <b>-l <i>datei</i></b> | ein symbolischer Link ( <b>l</b> ink) ist.                      |
| <b>-b <i>datei</i></b> | eine blockorientierte Datei ( <b>b</b> lock special) ist.       |
| <b>-c <i>datei</i></b> | eine zeichenorientierte Datei ( <b>c</b> haracter special) ist. |
| <b>-p <i>datei</i></b> | eine benannte Pipe (FIFO) ist ( <b>p</b> ipe).                  |

| Ausdruck               | liefert TRUE, wenn <i>datei</i> bzw. <i>n</i>                                                                                                            |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-S</b> <i>datei</i> | eine Socket-Datei ( <b>S</b> ocket) ist.                                                                                                                 |
| <b>-u</b> <i>datei</i> | Set-user-ID-Bit gesetzt ( <i>set-u</i> ser- <i>id</i> ) hat.                                                                                             |
| <b>-g</b> <i>datei</i> | Set-group-ID-Bit gesetzt ( <i>set-g</i> roup- <i>id</i> ) hat.                                                                                           |
| <b>-k</b> <i>datei</i> | Sticky-Bit gesetzt ( <i>stick</i> y) hat.                                                                                                                |
| <b>-t</b> <i>n</i>     | ein offener Filedeskriptor für ein Terminal ( <b>t</b> erminal) ist.                                                                                     |
| <b>-L</b> <i>datei</i> | (siehe unten)                                                                                                                                            |
| <b>-o</b> <i>datei</i> | dem Benutzer gehört ( <b>o</b> wnership).                                                                                                                |
| <b>-r</b> <i>datei</i> | gelesen ( <i>read</i> ) werden darf.                                                                                                                     |
| <b>-w</b> <i>datei</i> | beschrieben ( <b>w</b> rite) werden darf.                                                                                                                |
| <b>-x</b> <i>datei</i> | ausführbar ( <i>execute</i> ) ist.                                                                                                                       |
| <b>-X</b> <i>datei</i> | ausführbare <b>path</b> -Datei oder ein builtin-Kommando ist (z.B. liefert <b>-X ls</b> und <b>-X ls -F</b> TRUE, aber <b>-X /bin/ls</b> liefert FALSE). |
| <b>-z</b> <i>datei</i> | leer ist ( <b>z</b> ero).                                                                                                                                |
| <b>-s</b> <i>datei</i> | nicht leer ist ( <b>s</b> ize).                                                                                                                          |

Für die angegebene *datei* wird dabei Kommando-Substitution und Dateinamen-expandierung durchgeführt.

Die obigen Operatoren können auch kombiniert werden. So ist z.B. die Angabe

**-fx** *datei*

identisch zu

**-f** *datei* && **-x** *datei*

Beide würden 1 (TRUE) liefern, wenn *datei* eine normale Datei ist, die ausführbar ist. Für Directories dagegen, die ausführbar sind, liefert diese Angabe 0 (FALSE).

Der Operator **L** kann bei Angabe von mehreren Operatoren verwendet werden, um festzulegen, daß die nachfolgenden Operatoren bei symbolischen Links auf den Link selbst angewendet werden, und nicht auf die Datei, auf die der Link verweist. z.B. liefert **-Llo** *datei* TRUE (1), wenn es sich bei *datei* um symbolischen Link handelt, der dem aufrufenden Benutzer gehört. Die Angaben **-Lr**, **-Lw** und **-Lx** liefern für Links immer TRUE (1) und für alle anderen Dateien immer FALSE (0). **L** kann auch als letzter Operator angegeben werden. In diesem Fall hat **L** dann eine andere Bedeutung (siehe auch neue Operatoren weiter unten).

Alle Ausdrücke können mit den nachfolgenden Operatoren zu neuen Ausdrücken verknüpft werden:

!      Logischer Negationsoperator

&&    AND-Operator

||     OR-Operator

()    Klammern

Die folgende Tabelle zeigt die **Prioritätsreihenfolge** (höchste zuerst). Alle in einer Zeile angegebenen Operatoren besitzen dabei die gleiche Priorität:

()  
- (Minuszeichen)  
! ~  
\* / %  
+ -  
<< >>  
<= >= < >  
== != =~ !~  
&  
^  
|  
&&  
||

Alle Operatoren mit gleicher Priorität werden von links nach rechts abgearbeitet. Mit Klammerung ist es möglich, eine andere als die durch die Prioritäten vorgegebene Auswertung zu erzwingen.

Metazeichen wie \*, (, | usw. in Ausdrücken müssen nicht mit Quoting ausgeschaltet werden.

Die einzelnen Operanden und Operatoren eines Ausdrucks müssen durch Leerzeichen getrennt werden; dies gilt jedoch nicht für die Operatoren &, |, <, >, (, ).

Bei der Auswertung eines Ausdrucks werden Zahlen, die mit einer 0 beginnen, als Oktalzahlen interpretiert. Die Ergebnisse selbst werden jedoch immer im Dezimalsystem geliefert.

Fehlt ein Operand oder ist als Operand ein leerer String angegeben, setzt die tcsh hierfür den Wert 0 ein.

Wird der exit-Status eines Kommandos, das auszuführen ist, als Ausdruck benötigt, ist das entsprechende Kommando bzw. die betreffende Kommandoliste mit {..} zu klammern. { *kdoliste* } liefert 1, wenn die Ausführung von *kdoliste* erfolgreich (exit-Status 0) war, ansonsten liefert diese Konstruktion 0.

Wenn bei der Kompilierung der tcsh die Konstante **POSIX** definiert wurde (siehe auch Shellvariable **version**), ist das Ergebnis der obigen Datei-Operatoren nur von den entsprechenden Zugriffsrechten abhängig und nicht vom Rückgabewert der Systemfunktion *access*. Testet man z.B. mit **-w** eine Datei, die zwar das Schreibrecht gesetzt hat, sich aber auf einem Filesystem befindet, das als *read-only* montiert wurde, liefert diese Prüfung in einer POSIX-Shell TRUE und in einer anderen Shell FALSE.

Auf einigen tcsh-Versionen können die obigen Operatoren auch mit dem builtin-Kommando **filetest** ausgewertet werden.

Neben den obigen Operatoren, die immer TRUE (1) oder FALSE (0) liefern, bieten einige tcsh-Versionen weitere Operatoren, die andere Informationen liefern:

- A datei** Letzte Zugriffszeit als Sekundenzahl seit der Epoche (meist 0 Uhr am 1. Januar 1970).
- A: datei** wie **-A**, aber im Format »*Mon Jan 22 17:35:18 1996*«.
- M datei** Letzte Modifikationszeit als Sekundenzahl seit der Epoche (meist 0 Uhr am 1. Januar 1970).
- M: datei** wie **-M**, aber im Format »*Mon Jan 22 17:35:18 1996*«.
- C datei** Letzte Modifikationszeit des inode als Sekundenzahl seit der Epoche (meist 0 Uhr am 1. Januar 1970).
- C: datei** wie **-C**, aber im Format »*Mon Jan 22 17:35:18 1996*«.
- D datei** Gerätenummer von *datei*.
- I datei** Inode-Nummer von *datei*.
- F datei** Datei-Identifizierung im Format *geräten:inode*.
- L datei** Name der Datei, auf die ein symbolischer Link verweist.
- N datei** Anzahl von Hard-Links.
- P datei** Zugriffsrechte als Oktalzahl ohne führende Nullen.
- P: datei** wie **-P**, aber mit führenden Nullen.
- Pmodus datei** entspricht **-P datei & modus**; so liefert z. B. **-P21 datei** als Rückgabewert **21**, wenn *datei* von Gruppe beschreibbar und von den anderen ausführbar ist; Ist *datei* nur von der Gruppe beschreibbar, aber von den anderen nicht ausführbar, liefert dieser Ausdruck **20** als Rückgabewert. Obiger Ausdruck würde 0 liefern, wenn *datei* weder von der Gruppe beschreibbar noch von den anderen ausführbar ist.
- Pmodus: datei** wie **-Pmodus**, aber mit führenden Nullen.

|                         |                                                                        |
|-------------------------|------------------------------------------------------------------------|
| <b>-U</b> <i>datei</i>  | numerische user-ID.                                                    |
| <b>-U:</b> <i>datei</i> | Benutzername oder numerische user-ID, wenn Benutzername unbekannt ist. |
| <b>-G</b> <i>datei</i>  | numerische group-ID.                                                   |
| <b>-G:</b> <i>datei</i> | Gruppenname oder numerische group-ID, wenn Gruppenname unbekannt ist.  |
| <b>-Z</b> <i>datei</i>  | Größe in Bytes.                                                        |

Sind mehrere Operatoren auf einmal angegeben, kann nur einer der obigen Operatoren angegeben werden, der dann zusätzlich noch am Ende anzugeben ist. In diesem Fall hat der Operator **L** die in vorheriger Tabelle beschriebene Bedeutung, andernfalls die weiter oben beschriebene Bedeutung.

Da 0 ein erlaubter Rückgabewert bei vielen in der vorherigen Tabelle angegebenen Operatoren ist, geben diese Operatoren bei Fehler nicht **0**, sondern **-1** bzw. der Operator **-F** den Doppelpunkt (:) zurück.

## 8.8 Jobkontrolle

Die `tcsh` verfügt über eine eigene Jobkontrolle. Die einzelnen Jobs können dabei über die PID, die Jobnummer oder den Jobnamen angesprochen werden. Um einen Job über seine Jobnummer oder seinen Namen anzusprechen, gibt es folgende Notationen:

|                   |                                                                                                                        |
|-------------------|------------------------------------------------------------------------------------------------------------------------|
| <b>%jobnr</b>     | Job mit Jobnummer <i>jobnr</i>                                                                                         |
| <b>%string</b>    | Job, dessen Name mit <i>string</i> beginnt; <i>string</i> kann dabei Metazeichen der Dateinamenexpandierung enthalten. |
| <b>%?string</b>   | Job, dessen Name <i>string</i> enthält; <i>string</i> kann dabei Metazeichen der Dateinamenexpandierung enthalten.     |
| <b>%+ oder %%</b> | aktueller Job                                                                                                          |
| <b>%-</b>         | vorheriger aktueller Job                                                                                               |

Der Job, der gerade im Vordergrund abgearbeitet wird, kann mit der *Suspend*-Taste (meist **^Z**) angehalten werden.

Ist die Shellvariable **listjobs** gesetzt, werden bei **^Z** immer alle momentanen Jobs aufgelistet, so wie beim Aufruf des builtin-Kommandos **jobs**. Ist **listjobs** auf *long* gesetzt, werden dabei die Jobs wie bei **jobs -l** aufgelistet.

Ein Hintergrund-Job wird immer dann automatisch angehalten, wenn er versucht, vom Terminal zu lesen. Mit **stty tostop** kann zusätzlich noch festgelegt werden, daß Hintergrund-Jobs auch dann anzuhalten sind, wenn sie versuchen, auf das Terminal zu schreiben.



Für die Jobkontrolle stehen die builtin-Kommandos **jobs**, **kill**, **bg**, **fg**, **wait**, **stop**, **suspend** und **notify** zur Verfügung.

Das builtin-Kommando

**notify** [*job(s)*]

bewirkt z.B., daß die tcsh sofort eine Meldung ausgibt, wenn die Ausführung der angegebenen *job(s)* beendet ist oder sie blockiert werden. Normalerweise meldet die tcsh dies erst vor der Ausgabe des nächsten Prompts, um den Benutzer bei seiner Arbeit nicht durch ständige Meldungen zu stören. Werden keine *job(s)* beim Aufruf angegeben, bezieht sich **notify** auf den aktuellen Job (zuletzt im Hintergrund gestarteter Job oder angehaltener Job). Eine andere Möglichkeit ist, die Shellvariable **notify** zu definieren. In diesem Fall wird ebenfalls sofort eine Meldung ausgegeben, wenn ein Job sich beendet oder blockiert wird.

## 8.9 Periodische, automatische und zeitgesteuerte Ereignisse

Mit dem builtin-Kommando **sched** können Kommandos festgelegt werden, die von der tcsh zu einer bestimmten angegebenen Zeit ablaufen sollen.

Es existieren eine Reihe von speziellen Aliasnamen, mit denen Aktionen für bestimmte Ereignisse festgelegt werden können:

| <i>Alias</i>    | <i>die darin angegebenen Kommandos werden ...</i>                  |
|-----------------|--------------------------------------------------------------------|
| <b>beepcmd</b>  | immer dann ausgeführt, wenn die tcsh den Piepston ausgeben würde,  |
| <b>cwddcmd</b>  | immer dann ausgeführt, wenn das Working-Directory gewechselt wird, |
| <b>periodic</b> | alle <b>tperiod</b> -Minuten ausgeführt,                           |
| <b>precmd</b>   | immer vor der Ausgabe jedes Prompts ausgeführt.                    |

So bewirkt z.B. die Festlegung

**alias precmd date**

daß vor jeder Ausgabe des Prompts das aktuelle Datum angezeigt wird.

Die nachfolgenden Shellvariablen können entsprechend gesetzt werden, um bestimmte Aktionen auszuführen.

| Shellvariable          | kann gesetzt werden, um die <i>tcsh</i> anzuweisen, ...                                                                                                      |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>autologout</b>      | daß sie sich automatisch beendet oder sperrt, wenn die angegebenen Minuten verstrichen sind, ohne daß irgendwelche Eingaben erfolgten,                       |
| <b>mail</b>            | die Mailbox periodisch auf das Ankommen neuer Mail zu prüfen,                                                                                                |
| <b>printexit-value</b> | daß sie bei jedem nicht erfolgreichen Kommando den entsprechenden exit-Status ausgibt,                                                                       |
| <b>rmstar</b>          | daß sie bei jedem <b>rm *</b> automatisch nachfragt, ob dies wirklich erwünscht ist,                                                                         |
| <b>time</b>            | daß sie automatisch das <b>time</b> -Kommando nach der Beendigung von Kommandos ausführt, die eine gewisse vorgegebene CPU-Zeit (in Sekunden) überschreiten, |
| <b>watch, who</b>      | daß sie automatisch meldet, wenn sich bestimmte Benutzer an- oder abmelden.                                                                                  |

## 8.10 Abfangen von Signalen

Für asynchron eintreffende Interrupt- oder Beendigungssignale können mit dem builtin-Kommando **onintr** Signalhandler eingerichtet werden.

Zum Einrichten von Signalhandler in *tcsh*-Skripts steht das builtin-Kommando **onintr** zur Verfügung. Wird **onintr** ohne jegliche Argumente aufgerufen, wird die default-Signalbehandlung (*tcsh* beenden) für die entsprechenden Signale eingestellt. **onintr** - bewirkt, daß die entsprechenden Signale ignoriert werden und **onintr marke** bewirkt, daß beim Auftreten von Unterbrechungssignalen die Marke *marke* angesprungen wird.

Beim Lesen der Datei *~/logout* werden eintreffende Interrupt- und Beendigungssignale ignoriert. *quit*-Signale werden normalerweise ignoriert, außer die *tcsh* wurde mit der Option **-q** gestartet. Login-Shells fangen das *terminate*-Signal ab, und andere Shells erben den für *terminate* eingestellten Signalhandler von ihrem Elternprozeß. Bei allen anderen Signalen erben die Kindprozesse die eingestellten Signalhandler von ihrem Elternprozeß.

## 8.11 Builtin-Kommandos

Die builtin-Kommandos sind Teile des Programms *tcsh*. Deswegen hat die *tcsh* keinen neuen Prozeß zu starten, um diese ablaufen zu lassen, was zur Folge hat, daß diese Kommandos auch »schneller starten« als die anderen (nicht builtin-) Kommandos. Wenn jedoch builtin-Kommandos als Komponenten in einer Pipe angegeben sind, werden bis auf das letzte Kommando alle anderen Kommandos der Pipe in einer Subshell ausgeführt.

Die tcsh verfügt über folgende builtin-Kommandos:

**%job**

Kurzform zu **fg %job**

**%job &**

Kurzform zu **bg %job**

**:**

liefert immer erfolgreich (exit-Status 0); sonst keine Aktionen

**#**

Rest der Zeile wird als Kommentar interpretiert und somit ignoriert. tcsh-Skripts sollten immer mit einer Kommentar-Zeile beginnen. In einer interaktiven tcsh ist das Kommentarzeichen **#** nicht erlaubt.

**@** (1)

**@ variable = ausdruck** (2)

**@ variable[index] = ausdruck** (3)

Mit **@** ist es möglich, arithmetische Auswertungen durchzuführen und das Ergebnis dann Variablen bzw. Array-Elementen zuzuweisen:

- (1) gibt die Werte aller momentan definierten Shellvariablen aus,
- (2) setzt *variable* mit dem Ergebnis, das Auswertung von *ausdruck* liefert,
- (3) setzt das Array-Element *variable[index]* mit dem Ergebnis, das die Auswertung von *ausdruck* liefert. Hierbei ist zu beachten, daß *variable[index]* bereits existieren muß, also bereits zuvor mit **set** oder **setenv** deklariert sein muß.

Die Leerzeichen um den Zuweisungsoperator **=** sind bei (2) und (3) nicht notwendig. Für **=** können hierbei auch die von C her bekannten zusammengesetzten Operatoren **+=**, **-=**, **\*=**, **/=**, **%=** und **^=** verwendet werden. Auch ist die Verwendung der Postfix-Operatoren **++** (Inkrementieren) und **--** (Dekrementieren) erlaubt:

**@ variable++** bzw. **@ variable--**

**@ variable[index]++** bzw. **@ variable[index]--**

Im *ausdruck* können die meisten von C her bekannten Operatoren verwendet werden. Dabei ist lediglich zu beachten, daß bei Verwendung einer der Operatoren **<**, **>**, **&** oder **|** der *ausdruck* oder zumindest der entsprechende Teilausdruck mit **(..)** zu klammern ist.

**alias** [*name*] [*wortliste*]

Wird **alias** alleine ohne jegliche Argumente aufgerufen, dann gibt es eine Liste aller definierten Aliase auf die Standardausgabe aus.

Wird **alias** mit einem *namen* (aber ohne Angabe einer *wortliste*) aufgerufen, zeigt es den Wert des Aliases *name* an.

Wird bei einem Aufruf von **alias** ein *name* und eine *wortliste* angegeben, wird ein Alias *name* definiert, dem als Wert diese *wortliste* zugewiesen wird. Für *wortliste* findet dabei Kommando-Substitution und Dateinamenexpansion statt. Für *name* darf niemals **alias** oder **unalias** angegeben werden.

**alloc**

Zeigt den momentan freien und benutzten Speicher an.

**bg** [%*job(s)*]

Setzt die Ausführung von angehaltenen *job(s)* im Hintergrund fort. Sind keine *job(s)* angegeben, wird die Ausführung des aktuellen Jobs (zuletzt angehaltener Job) im Hintergrund fortgesetzt. Für *job* kann eine Nummer, ein String, %, + oder - angegeben werden (siehe auch Jobkontrolle in Kapitel 8.8)

**bindkey** [-l | -d | -e | -v | -u]**bindkey** [-a][-b][-k][-r][--] *tastenkombi***bindkey** [-a][-b][-k][-c | -s][--] *tastenkombi editor-kommando***bindkey** [ohne Optionen oder sonstige Argumente]

Gibt alle voreingestellten Tastenkombinationen mit den entsprechenden Editor-Kommandos aus.

**bindkey** *tastenkombi*

Gibt das Editor-Kommando aus, das der Tastenkombination *tastenkombi* zugeordnet ist.

**bindkey** *tastenkombi editor-kommando*

Bindet das *editor-kommando* an die Tastenkombination *tastenkombi*,

- l** (**list**) Listet alle möglichen Editor-Kommandos mit einer kurzen Beschreibung
- d** (**default**) Vorgegebene Tastenkombinationen für den default-Kommandozeilen-Editor einstellen
- e** (**emacs**) Vorgegebene Tastenkombinationen für Kommandozeilen-Editor emacs einstellen
- v** (**vi**) Vorgegebene Tastenkombinationen für Kommandozeilen-Editor vi einstellen

- a (**a**lternative) Auflisten oder Ändern der alternativen *key-bindings*. Dies sind die *key-bindings* im *vi*-Kommandomodus.
- b **^**zeichen (wie **^A**) bzw. **C**-zeichen (wie **C-A**) wird als entsprechendes *Strg*-Zeichen, **M**-zeichen (wie **M-A**) wird als entsprechendes Meta-Zeichen, **F**-string wird als entsprechende Funktionstaste und **X**-zeichen (wie **X-A**) wird als entsprechende erweiterte Präfix-Taste interpretiert.
- k (**k**ey) *tastenkombi* wird als Pfeil-Taste interpretiert. Für *tastenkombi* kann dabei **down**, **up**, **left** oder **right** angegeben werden.
- r (**r**emove) Löscht die eingestellte *tastenkombi*.
- c (**c**ommand) Das angegebene *editor-kommando* wird nicht als Editor-Kommando, sondern als Unix-Kommando interpretiert.
- s (**s**tring) Das angegebene *editor-kommando* wird als String interpretiert, der an Cursorposition in der Kommandozeile einzufügen ist, wenn *tastenkombi* eingegeben wird. Enthält der angegebene String Editor-Kommandos, werden diese beim Einfügen ausgeführt. Dabei werden rekursiv definierte Editor-Kommandos bis zur Rekursionstiefe 10 ausgewertet.
- Beendet die Optionen-Angabe, so daß das nächste Argument als *tastenkombi* interpretiert wird, selbst wenn es mit Bindestrich (-) beginnt.
- u (**u**sage) Gibt Syntax des **bindkey**-Kommandos aus.

*sonstige unerlaubte Option* Gibt Syntax des **bindkey**-Kommandos aus.

In *tastenkombi* oder *editor-kommando* dürfen folgende Escape-Sequenzen angegeben werden:

- \a Piepston
- \b Backspace
- \e ESC
- \f Seitenvorschub (*Form feed*)
- \n Neuezeile-Zeichen (*Newline*)
- \r Wagenrücklauf (*Carriage Return*)
- \t Horizontales Tabzeichen
- \v Vertikales Tabzeichen
- \nnn ASCII-Zeichen mit dem oktalen ASCII-Code *nnn*

Um die Sonderbedeutung von Zeichen wie **^** oder **\** auszuschalten, muß ihnen ein Backslash (**\**) vorangestellt werden.

**break**

**break** bewirkt das Verlassen einer umschließenden **while**- oder **foreach**-Schleife. Da die restlichen Kommandos in einer **break**-Zeile noch vor dem Verlassen der entsprechenden Schleife ausgeführt werden, können mehrere **break** gegeben werden: **break; break; ...**, wenn mehrere Schleifenebenen zugleich zu verlassen sind.

**breaksw**

**breaksw** bewirkt das Verlassen einer **switch**-Anweisung.

**builtins**

gibt die Namen aller builtin-Kommandos aus.

**bye**

Ist ein Synonym für das builtin-Kommando **logout**. **bye** ist nur verfügbar, wenn die tcsh entsprechend kompiliert wurde (siehe auch Shellvariable **version**).

**case** *marke*:

Legt eine Marke für die **switch**-Anweisung fest.

**cd** [-p][-l][-n|-v] [*directory*]

**chdir** [-p][-l][-n|-v] [*directory*]

**cd** bzw. **chdir** bewirkt einen Wechsel zum angegebenen *directory*. Ist *directory* als absoluter Pfadname angegeben, wird dorthin gewechselt, wenn möglich. Existiert *directory* nicht als Subdirectory im Working-Directory oder beginnt der dafür angegebene Pfadname nicht mit **/**, **./** oder **../**, wird in den von **cdpath** festgelegten Directories nach diesem Pfadnamen gesucht, um dorthin zu wechseln. Ist kein *directory* angegeben, wird zum **home**-Directory gewechselt.

Wird für *directory* ein Bindestrich (**-**) angegeben, wird zum vorherigen Working-Directory gewechselt.

**-p** gibt wie **dirs** den Directorystack aus. Die Optionen **-l**, **-n** und **-v** haben die gleiche Wirkung wie beim Kommando **dirs** und implizieren die Option **-p**.

**complete** [*kommando* [*wort/pattern/liste[:auswahl]*] / [[*suffix*]/]...]]

Sind keinerlei Argumente angegeben, werden alle mit **complete** eingestellten Vervollständigungsmöglichkeiten ausgegeben.

Ist *kommando* angegeben, wird die für *kommando* eingestellte Vervollständigung ausgegeben.

In allen anderen Fällen werden Vervollständigungsmöglichkeiten definiert.

Für *kommando* kann sowohl ein voller Kommandoname oder ein *pattern* angegeben werden (siehe auch Dateinamen-Substitution in Kapitel 8.5.4). Beginnt dieser Kommandoname oder das *pattern* mit einem Bindestrich (-), soll Vervollständigung nur angewendet werden, wenn das *kommando* mehrdeutig ist.

*wort* legt fest, welches Wort relativ zum aktuellen Wort zu vervollständigen ist. Für *wort* kann dabei folgendes angegeben werden:

- c** (**c**urrent word) Vervollständigung für aktuelles Wort. *pattern* muß dabei den Anfang des aktuellen Wortes in der Kommandozeile abdecken. *pattern* wird bei Vervollständigung des aktuellen Wortes ignoriert.
- C** (**C**urrent word) Wie **c**, nur daß bei Vervollständigung des aktuellen Wortes *pattern* nicht ignoriert wird.
- n** (**n**ext word) Vervollständigung für nächstes Wort. *pattern* muß dabei den Anfang des vorherigen Wortes in der Kommandozeile abdecken.
- N** (**N**ext word) Wie **n**, nur daß *pattern* dabei den Anfang des vor-vorherigen Wortes abdecken muß.
- p** (**p**osition dependent) Positionsabhängige Vervollständigung. *pattern* muß dabei eine Bereichsangabe sein, die dem Index bei Zugriff auf bestimmte Wörter in einer Shellvariable entspricht. Das aktuelle Wort muß sich dabei innerhalb dieses Bereiches befinden.

Für *liste*, womit die möglichen Vervollständigungen festgelegt werden, kann folgendes angegeben werden:

- a** Aliase
- b** Bindings (Editor-Kommandos)
- c** Kommandos (builtin oder sonstige)
- C** Nicht-builtin-Kommandos, die mit dem angegebenen Pfad beginnen
- d** Directories
- D** Directories, die mit dem angegebenen Pfad beginnen
- e** Environment-Variablen
- f** Dateinamen (*file names*)
- F** Dateinamen (*File names*), die mit dem angegebenen Pfad beginnen
- g** Gruppennamen (*groupnames*)
- j** Jobs
- l** Limits

|              |                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------|
| <b>n</b>     | Nichts                                                                                           |
| <b>s</b>     | Shellvariablen                                                                                   |
| <b>S</b>     | Signale                                                                                          |
| <b>t</b>     | Echte »text«-Dateien                                                                             |
| <b>T</b>     | Echte »text«-Dateien, die mit dem angegebenen Pfad beginnen                                      |
| <b>u</b>     | Benutzernamen ( <i>usernames</i> )                                                               |
| <b>v</b>     | Beliebige Variablen                                                                              |
| <b>x</b>     | wie <b>n</b> , aber gibt <i>auswahl</i> bei Eingabe des Editor-Kommandos <i>list-choices</i> aus |
| <b>X</b>     | Vervollständigungen                                                                              |
| <b>\$var</b> | Wörter aus der Variable <i>var</i>                                                               |
| <b>(...)</b> | Wörter aus der angegebenen Kommandoliste                                                         |
| <b>`...`</b> | Wörter aus der Ausgabe des in <b>`...`</b> angegebenen Kommandos (Kommando-Substitution)         |

Für *auswahl*, das optional ist, muß ein *pattern* angegeben werden. Ist *auswahl* angegeben, werden nur die Wörter aus *liste* betrachtet, die durch dieses *pattern* abgedeckt sind. Die Shellvariable **ignore** wird ignoriert. Für die letzten drei Angaben aus der obigen Tabelle ist keine *auswahl* erlaubt, und **x** gibt bei Eingabe des Editor-Kommandos *list-choices* die angegebene *auswahl* als erklärende Meldung aus.

Für *suffix* ist ein einfaches Zeichen anzugeben, das bei einer erfolgreichen Vervollständigung angehängt wird. Ist für *suffix* nichts angegeben, wird kein Zeichen angehängt. Wird die *suffix*-Angabe vollständig weggelassen, wird an Directories ein Slash (/) und an alle anderen Wörter ein Leerzeichen angehängt.

Nachfolgend werden einige Beispiele zu der Vielzahl von möglichen Angaben bei **complete** gegeben:

Manchmal ist es wünschenswert, daß man eine Vervollständigung von Wörtern nur auf Directories begrenzt, wie z.B. beim Kommando **cd**:

```
> complete cd 'p/1/d/'
> ls -F b*
bin/ brief1.txt brief2.txt
> cd b[TAB-Taste]
> cd bin/
```

Hier wurde die Vervollständigung eines Wortes, das nach **cd** angegeben ist (p/1), auf Directories (/d) eingeschränkt.



Mit **p** kann auch die Auswahl von möglichen Kommandovervollständigungen eingeschränkt werden, wie z.B.

```
> ba[^D]
badblocks banner basename bash batch
> complete -ba* 'p/0/(basename)/'
> ba[TAB-Taste]
> basename
```

Hier wurde festgelegt, daß Kommandos (Wörter an Position 0; p/0), die mit **ba** beginnen, zu **basename** (einziges Wort in der Liste) zu vervollständigen sind. Der führende Bindestrich (-) legt fest, daß diese Vervollständigung nur bei Kommandos zu verwenden ist, die mehrere Vervollständigungen erlauben.

Das nächste ist ein Beispiel zur **n**-Angabe:

```
> complete find 'n/-user/u/'
```

Jedes Wort in einem **find**-Kommando, das unmittelbar der Angabe **-user** folgt, wird zu einem Loginnamen vervollständigt.

Das nächste ist ein Beispiel zur **c**-Angabe:

```
> complete cc 'c/-I/d/'
```

Jedes Wort in einem **cc**-Aufruf, das mit **-I** beginnt, wird zu einem Directorynamen vervollständigt. Da kleines **c** verwendet wurde, wird **-I** nicht als Teil des Directorynamens betrachtet.

Es macht Sinn, unterschiedliche *liste*-Angaben für verschiedene Kommandos festzulegen.

```
> complete alias 'p/1/a/'
> complete man 'p/*/c/' [auch mögl. 'n/*/c/' oder 'c/*/c/']
> complete set 'p/1/s/'
> complete true 'p/1/x:true hat keine Optionen/'
```

Obige Angaben bewirken folgendes:

- ▶ Wörter, die **alias**-Kommando folgen, werden zu Aliasnamen vervollständigt.
- ▶ Wörter, die **man**-Kommando folgen, werden zu Kommandonamen vervollständigt.
- ▶ Wörter, die **set**-Kommando folgen, werden zu Shellvariablen vervollständigt.
- ▶ Bei dem Versuch **true** zu vervollständigen, wird »true hat keine Optionen« ausgegeben.

Wird in der *liste*-Angabe auf eine Shellvariable zugegriffen, wird diese erst zum Zeitpunkt der Vervollständigung ausgewertet:

```
> complete mailx 'p/1/$gruppe/'
> set gruppe = (hans fritz claudia)
> mailx [^D]
claudia fritz hans
> mailx [^C]
> set gruppe = (gunter toni micha)
> mailx [^D]
gunter toni micha
```

Wird in der *liste*-Angabe ein Kommando (in `...`) angegeben, wird diese Kommando-Substitution erst zum Zeitpunkt der Vervollständigung ausgeführt:

```
> complete mount 'p.*.`cat /etc/fstab | awk \{print\ \$2\}`.'
> mount [^D]
/ /A /B /C /D /cdrom /proc
```

Damit die Sonderzeichen \$, Leerzeichen, { und } nicht von der Shell ausgewertet, sondern an awk weitergeleitet werden, müssen sie durch das Voranstellen eines Backslash (\) ausgeschaltet werden. Auch ist an obigem Beispiel erkennbar, daß anstelle von Slash (/) zum Trennen von *wort*, *pattern* und *liste* jedes beliebige andere Zeichen verwendet werden kann (im Beispiel ein Punkt). Dies ist im obigen Beispiel notwendig, da der Slash als Teil des Pfadnamens */etc/fstab* vorkommt.

Für ein Kommando können mehr als eine Vervollständigung angegeben werden:

```
> complete dbx 'p/2/(core)/' 'p/*/c/'
```

Mit dieser Angabe wird festgelegt, daß das zweite Wort zu **dbx** zum Wort *core* und alle anderen Argumente zu Kommandos vervollständigt werden. Es ist dabei die positionsabhängige Angabe vor der **n**-Angabe angegeben. Der Grund dafür ist, daß die Vervollständigungen von links nach rechts ausgewertet werden. Wenn nun die **n**-Angabe vor der **p**-Angabe stünde, würde sie bereits alle Worte abdecken und die **p**-Angabe würde niemals ausgewertet.

Die Verwendung von *auswahl* ist hilfreich, wenn Kommandos nur bestimmte Dateien als Argumente erwarten, wie z.B.:

```
> complete cc 'p/*/f:*. [coa]'
```

Argumente bei einem **cc**-Aufruf werden nur zu Dateien vervollständigt, die mit *.c*, *.o* oder *.a* enden. Mit der Angabe des Negationsoperators **^** können über *auswahl* auch bestimmte Dateien von der Vervollständigung ausgeschlossen werden, wie z.B.:

```
> complete rm 'p/*/f:^^.{c,h,y,C,man,1}'
```

Die *liste*-Angaben **C**, **D**, **F** und **T** verhalten sich weitgehend wie **c**, **d**, **f** und **t**, nur daß sie das *auswahl*-Argument nur auf Dateien anwenden, die mit dem angegebenen Pfad beginnen. Zum Beispiel verwendet das mail-Programm **elm** das Zeichen **=** als Abkürzung für das eigene mail-Directory:

```
> complete elm 'c+=F:$HOME/post/+'
```

Der Aufruf **elm -f =** würde nun zu **elm -f ~/post/** vervollständigt. Hier wurde anstelle des Trennzeichens **/** das Trennzeichen **+** verwendet, um nicht mit den Slashes aus dem Pfadnamen **\$HOME/post/** in Konflikt zu geraten. Auch wurde **\$HOME** anstelle von **~** benutzt, da Tilde-Substitution nur am Beginn eines Wortes durchgeführt wird.

Die *suffix*-Angabe kann verwendet werden, um festzulegen, daß eigene Suffixes (nicht Leerzeichen oder **/** bei Directories) an vervollständigte Wörter angehängt werden.

```
> complete finger 'c/*@/$hostnames/' 'p/1/u/@'
```

Diese Angabe vervollständigt die Argumente bei einem **finger**-Aufruf zu einer Liste von Benutzernamen, hängt diesen dann ein **@** an, bevor es die entsprechenden Hostnamen anhängt. Hier spielt die Reihenfolge der Angabe wieder eine Rolle.

Mit **complete** eingestellte Vervollständigungen haben keine Wirkung, wenn ein Wort mit **~** (Tilde) oder **\$** (Variable) beginnt. Um eine oder mehrere mit **complete** eingestellte Vervollständigungen wieder zu löschen, steht das builtin-Kommando **uncomplete** zur Verfügung.

## continue

startet unmittelbar den nächsten **while** oder **foreach**-Schleifendurchlauf. Da die restlichen Kommandos in einer **continue**-Zeile noch vor dem Neustart des nächsten Schleifendurchlaufs ausgeführt werden, können mehrere **continue** gegeben werden: **continue; continue; ...**, wenn ein nächster Schleifendurchlauf für eine weiter außen liegende Schleife zu starten ist.

## default:

ist die default-Marke in einer **switch**-Anweisung. Sie sollte als letzte aller switch-Marken angegeben sein.

**dirs** [-l] [-n | -v] (1)

**dirs** -S | -L [dateiname] (2)

**dirs** [-c] (3)

- (1) gibt den Directorystack von oben nach unten in einer Zeile aus. Das erste Directory ist dabei immer das Working-Directory.

Ist die Option **-l** angegeben, wird für die Ausgabe Tilde-Substitution durchgeführt und anstelle von ~ der entsprechende Pfadname ausgegeben.

Ist die Option **-n** angegeben, wird bei der Ausgabe automatisch vor Erreichen des Zeilenendes ein Zeilenumbruch durchgeführt.

Ist die Option **-v** angegeben, wird jeder Eintrag des Directorystacks in einer eigenen Zeile mit einer vorangestellten Nummer (Position im Directorystack) ausgegeben.

Sind mehr als ein **-n** oder **-v** angegeben, so hat **-v** Vorrang.

- (2) Ist die Option **-S** angegeben, wird der Directorystack als eine Folge von **cd**- und **pushd**-Kommandos in der Datei *dateiname* gesichert.

Ist die Option **-L** angegeben, liest die tcsh den Inhalt der Datei *dateiname* und führt ihn aus. *dateiname* sollte dabei eine Datei sein, die zuvor mit Option **-S** oder mit dem **savedirs**-Mechanismus erstellt wurde.

Ist kein *dateiname* angegeben, wird in beiden Fällen die in der Shellvariable **dirsfile** angegebene Datei benutzt. Sollte auch **dirsfile** nicht gesetzt sein, wird die Datei *~/.cshdirs* zum Schreiben (**-S**) bzw. zum Lesen (**-L**) verwendet. Login-Shells führen beim Aufruf **dirs -L** aus. Wenn **savedirs** gesetzt ist, führen sie beim Abmelden **dirs -S** aus. Da normalerweise beim Anmelden nur *~/.tcshrc* vor *~/.cshdirs* gelesen wird, sollte **dirsfile** in *~/.tcshrc* und nicht in *~/.login* gesetzt werden.

- (3) **dirs -c** löscht den Inhalt des Directorystacks.

**echo** [-n] [argument(e)]

gibt die *argument(e)*, nachdem für diese Variablen-, Kommando- und Dateinamen-Substitution durchgeführt wurde, auf die Standardausgabe aus. Bei der Ausgabe werden alle Argumente durch ein Leerzeichen voneinander getrennt. Diese gesamte Ausgabe wird dann mit einem Neuezeile-Zeichen abgeschlossen. Ist die Option **-n** angegeben, wird die Ausgabe nicht mit einem Neuezeile-Zeichen abgeschlossen. Es kann die Variable **echo\_style** entsprechend gesetzt werden, damit **echo** sich z.B. wie unter BSD oder unter System V verhält.

**echotc** [-sv] *argument(e)*

*tc* steht dabei für *terminal-capabilities* (siehe auch Man-page zu **termcap**). z.B. positioniert die folgende Kommandozeile den Cursor links oben am Bildschirm:

**echotc home**

und

**echotc cm 7 15**

positioniert den Cursor in der 7. Spalte und 15. Zeile des Bildschirms. Die folgende Kommandozeile

**echotc mr; echo "Diese Zeile ist im reverse-modus"; echotc me**

würde den Text *Diese Zeile ist im reverse-modus* im reverse-Modus (Hintergrund weiß; Schrift in schwarz) ausgeben.

Ist für die *argument(e)* **baud**, **cols**, **lines**, **meta** oder **tabs** angegeben, wird der Wert für diese *capability* ausgegeben (Ausgabe *yes* oder *no* zeigt an, ob der Terminal diese *capability* unterstützt oder nicht). Hiermit kann man z.B. die Zeilenausgabe für **history** heruntersetzen:

```
> set history = `echotc lines`
> echo $history
25
> @ history--
> @ history--
> echo $history
23
```

Beim Setzen von Variablen mit *termcap*-Strings sollten grundsätzlich Anführungszeichen verwendet werden, wie z.B.:

```
> set tosl = "`echotc mr`"
> set frsl = "`echotc me`"
> echo -n "$tosl"; who; echo -n "$frsl"
```

Die letzte Kommandozeile bewirkt, daß alle momentan angemeldeten Benutzer mit **who** im reverse-Modus ausgeben werden.

Ist Option **-s** angegeben, liefern nicht existierende *capabilities* den Leerstring und führen nicht zu einem Fehler. Bei Angabe von **-v** werden ausführlichere Fehlermeldungen ausgegeben.

**else**  
**end**  
**endif**  
**endsw**

siehe Beschreibung von **foreach**, **if**, **switch** und **while**.

**eval** *argument(e)*

liest und wertet eine Kommandozeile zweimal aus, bevor sie ausgeführt wird.

**exec** *argument(e)*

*argument(e)* spezifizieren ein Kommando mit seinen Argumenten, das anstelle der momentan aktiven tcsh ausgeführt wird.

**exit** [*ausdr*]

Ist *ausdr* (siehe auch tcsh-Ausdrücke in Kapitel 8.7) nicht angegeben, beendet sich die aktive tcsh mit dem momentanen Wert der Shellvariablen **status** als exit-Status, ansonsten beendet sie sich mit dem Wert, den der Ausdruck *ausdr* liefert, als exit-Status.

**fg** [%*job(s)*]

**fg** setzt die Ausführung von angehaltenen *job(s)* im Vordergrund fort. Sind keine *job(s)* angegeben, wird die Ausführung des aktuellen Jobs (zuletzt angehaltener Job) im Vordergrund fortgesetzt. Für *job* kann eine Nummer, ein String, %, + oder - angegeben werden (siehe auch Jobkontrolle in Kapitel 8.8 und Editor-Kommando *run-fg-editor* in Kapitel 8.3.4).

**filetest** -*op* *datei(en)*

wendet den Operator *op* auf die angegebenen *datei(en)* an und liefert die Ergebnisse mit Leerzeichen getrennt in Form einer Liste (siehe auch tcsh-Ausdrücke in Kapitel 8.7).

**foreach** *variable* (*wortliste*)

*kdos*

**end**

*variable* wird nacheinander mit den einzelnen Worten der angegebenen *wortliste* besetzt, und jedesmal werden die *kdos* ausgeführt. Die Schlüsselwörter **foreach** und **end** müssen als erstes Wort einer Zeile angegeben sein. Wird die **foreach**-Schleife interaktiv eingegeben, zeigt die tcsh, solange nicht das abschließende **end** eingegeben wurde, mit der Ausgabe des Prompts *foreach?* oder dem Inhalt von **prompt2** an, daß die **foreach**-Schleife noch nicht abgeschlossen ist.

**getspath** (nur auf TCF)

gibt den momentanen System-Ausführungspfad aus.

**Kommando: getxvers** (nur auf TCF)

gibt das experimentelle Versions-Präfix aus.

**glob** [*argument(e)*]

entspricht weitgehend dem Kommando **echo**, nur daß Escape-Sequenzen (**\n**, **\c**) ihre Sonderbedeutung verlieren und die einzelnen *argument(e)* bei der Ausgabe nicht durch Leerzeichen getrennt, sondern wie in C mit **\0** abgeschlossen werden; auch wird die Ausgabe nicht mit einem Neueilen-Zeichen abgeschlossen.

**goto** *wort*

bewirkt einen Sprung auf die mit *wort* ausgewählte Marke. Diese Marke muß mit

*marke:*

angegeben sein. Für *wort* wird Dateinamen- und Kommando-Substitution durchgeführt.

**hashstat**

gibt eine Statistik aus, die über den Erfolgsgrad der internen Hashing-Tabelle informiert. Diese Statistik zeigt an, wie viele aufgerufene Kommandos auch wirklich gefunden wurden und wie viele nicht; dazu wird noch eine Prozentzahl angegeben.

**history** [-hr] [*n*] (1)

**history** -S | -L | -M [*dateiname*] (2)

**history** -c (3)

(1) Wird **history** ohne jegliche Argumente aufgerufen, gibt es alle in der History-Liste aufgehobenen Kommandozeilen aus.

Wird **history** mit einer Zahl *n* aufgerufen, gibt es nur die letzten *n* Kommandozeilen aus der History-Liste aus.

Wird **history** mit der Option **-r** aufgerufen, gibt es die in der History-Liste aufgehobenen Kommandozeilen in umgekehrter Reihenfolge aus.

Wird **history** mit der Option **-h** aufgerufen, gibt es die in der History-Liste aufgehobenen Kommandozeilen ohne führende Nummern und Zeitmarken (*timestamps*) aus. Dies kann benutzt werden, um Dateien zu erzeugen, die dann später mit **history -L** oder **source -h** wieder geladen werden können.

(2) Ist die Option **-S** angegeben, wird die History-Liste in der Datei *dateiname* gesichert. Ist das erste Wort der Shellvariable **savehist** eine Zahl, werden nur so viele Kommandozeilen aus der History-Liste in der Datei gesichert. Ist das zweite Wort von **savehist** *merge*, werden die Kommandozeilen mit den eventuell schon in der Datei vorhandenen Kommandozeilen gemischt, wobei sie entsprechend den Zeitmarken sortiert werden. Ein solches Mischen ist nützlich in einer Umgebung wie X Windows oder beim Arbeiten mit virtuellen Terminals, wo meist mehrere Shells gleichzeitig aktiv sind.

Ist die Option **-L** angegeben, hängt die Shell den Inhalt der Datei *dateiname* (mit **-S** oder über den **savehist**-Mechanismus erzeugt) an die aktuelle History-Liste an. Die Option **-M** ist ähnlich zu **-L**, nur daß der Inhalt von *dateiname* nicht an die History-Liste angehängt, sondern – entsprechend den Zeitmarken – in sie einsortiert wird. Sowohl bei **-L** als auch bei **-M** wird **histfile** verwendet, wenn *dateiname* nicht angegeben ist. Ist auch **histfile** nicht gesetzt, wird *~/.history* verwendet.

**history -L** ist im übrigen identisch zum Aufruf **source -h**, nur daß bei **history -L** nicht unbedingt ein Dateiname angegeben sein muß.

Login-Shells laden beim Start die History-Liste (wie **history -L**) und, falls **savehist** gesetzt ist, sichern sie die aktuelle History-Liste beim Beenden (wie **history -S**).

Da normalerweise beim Anmelden nur *~/.tcshrc* vor *~/.history* gelesen wird, sollte **histfile** in *~/.tcshrc* und nicht in *~/.login* gesetzt werden.

(3) **history -c** löscht den Inhalt der History-Liste.

Ist die Shellvariable **histlit** gesetzt, wird bei den Aufrufformen (1) und (2) die History-Liste in der unexpandierten Form ausgegeben bzw. gesichert.

**hup** [*kommando*]

Wird **hup** in einem Skript ohne ein Argument aufgerufen, wird bei einem nachfolgenden Auftreten eines *hangup*-Signals die Skript-Ausführung beendet. Wird **hup** mit einem *kommando* aufgerufen, wird das *kommando* ausgeführt. Sollte jedoch während der Ausführung ein *hangup*-Signal auftreten, wird dieses *kommando* sofort beendet. Das Gegenstück zu **hup** ist das builtin-Kommando **nohup**.

**if** (*ausdruck*) *kommando*

Wenn die Auswertung von *ausdruck* (siehe auch tcsh-Ausdrücke in Kapitel 8.7) TRUE liefert, wird das angegebene *kommando*, das sich in der gleichen Zeile befinden muß, ausgeführt, ansonsten wird es nicht ausgeführt. *kommando* muß ein einfaches Kommando sein; es darf also kein Alias, keine Pipeline, Kommandoliste oder geklammerte Kommandoliste dafür angegeben werden. Vorsicht: Ein- und Ausgabeumlenkungen werden auch dann durchgeführt, wenn die Ausführung von Kommando nicht erfolgreich war.

**if** (*ausdr1*) **then**

*then\_kdoliste1*

**else if** (*ausdr2*) **then**

*then\_kdoliste2*

:

**[ else**

*else\_kdoliste* ]

**endif**



Wenn die Auswertung von *ausdr1* TRUE liefert, wird die *then\_kdolist1* ausgeführt, ansonsten wird *ausdr2* ausgewertet. Liefert dieser TRUE, wird *then\_kdolist2* ausgeführt usw.. Liefern alle angegebenen Ausdrücke FALSE, wird die *else\_kdolist* ausgeführt, wenn **else** angegeben ist. Das Schlüsselwort **if** muß entweder als erstes Wort einer Zeile oder nach **else** angegeben sein. Das Schlüsselwort **then** muß in der gleichen Zeile wie **if** (*ausdr*) stehen. Die Schlüsselwörter **else** und **endif** müssen als erstes Wort einer Zeile angegeben sein.

**inlib** *shared-library* ... (nur in Domain / OS)

fügt jede angegebene *shared-library* zur aktuellen Environment hinzu. Ein nachträgliches Entfernen ist nicht mehr möglich.

**jobs** [-l]

gibt Informationen zu aktiven Jobs aus. Ist die Option **-l** angegeben, wird zusätzlich zur normalen Information noch die PID angezeigt.

**kill** [-signal] %job | pid ... (1)

**kill -l** (2)

Mit der ersten Aufrufform (1) wird den angegebenen *jobs* oder Prozessen (*pid*) das Signal *signal* geschickt. Dieses geschickte Signal kann von den entsprechenden *jobs* oder Prozessen entweder ignoriert oder mit einem installierten Signalhandler abgefangen werden, andernfalls werden die betreffenden *jobs* oder Prozesse durch diesen Aufruf abgebrochen. Für *signal* kann entweder eine Signalnummer oder ein Signalname angegeben werden. Ist *-signal* nicht angegeben, dann wird das Signal **TERM** geschickt. Für *job* kann eine Nummer, ein String, %, + oder - angegeben werden (siehe auch Jobkontrolle in Kapitel 8.8).

Um sich alle auf dem jeweiligen System verfügbaren Signalnummern und Signalnamen anzeigen zu lassen, steht die zweite Aufrufform (2) zur Verfügung: **kill -l**

**limit** [-h] [*resource* [*maximum*]]

Mit **limit** können Limits für Systemressourcen festgelegt bzw. angezeigt werden. **limit** legt ein maximales Limit (*maximum*) für die angegebene *resource* fest. Dieses Limit darf vom aktuellen Prozeß und allen seinen Kindprozessen nicht überschritten werden.

Ist kein *maximum* angegeben, wird das momentan festgelegte Limit für *resource* oder für alle Ressourcen, wenn *resource* fehlt, ausgegeben. Ist die Option **-h** angegeben, werden anstelle der momentanen Limits die festgelegten Hard-Limits ausgegeben. Hard-Limits sind die vom Superuser festgelegten Limits, die kein Benutzer hochsetzen kann. Ein Erhöhen oder Erniedrigen der momentanen Limits (Soft-Limits) ist dagegen jedem Benutzer erlaubt, solange er nicht das Hard-Limit überschreitet.

Für *resource* kann dabei folgendes angegeben werden: **cputime**, **filesize**, **datasize**, **stacksize**, **coredumpsize**, **memorysize**. Diese Schlüsselwörter können auch abgekürzt angegeben werden, solange sie eindeutig sind.

Für *maximum* kann eine Gleitpunktzahl oder eine ganze Zahl angegeben werden. Für alle Limits außer **cputime** wird dabei **k** als Suffix für Kilobytes (1024 Bytes) angenommen. Möchte der Benutzer Megabytes angeben, muß er hinter der entsprechenden Zahl ein **m** angeben. Für **cputime** legt die angegebene Zahl Sekunden fest. Ist nach der Zahl **m** angegeben, werden Minuten bzw. mit **h** Stunden festgelegt. Es kann auch bei **cputime** anstelle einer Zahl das Format *mm:ss* angegeben werden, wobei *mm* die Minuten- und *ss* die Sekundenanzahl festlegt.

### **log**

gibt alle Benutzer aus, die in der Variable **watch** angegeben und momentan auch angemeldet sind (siehe auch Variable **watchlog**).

### **login** [*loginname*]

beendet die Login-Shell und startet die Login-Prozedur für eine neue Unix-Sitzung; wird ein *loginname* angegeben, wird sofort nach dem Paßwort des entsprechenden Benutzers gefragt.

### **logout**

beendet eine Login-Shell; nützlich wenn **ignoreeof** gesetzt ist.

### **ls-F** [-switch ...] [*datei(en)*]

listet Dateien wie **ls -F** auf, jedoch wesentlich schneller. Bei der Ausgabe werden die Dateinamen durch Anhängen eines der folgenden Zeichen entsprechend klassifiziert:

|   |                                                                     |
|---|---------------------------------------------------------------------|
| / | Directory                                                           |
| * | Ausführbare Datei                                                   |
| # | Blockorientierte Gerätedatei                                        |
| % | Zeichenorientierte Gerätedatei                                      |
|   | Benannte Pipe ( <i>Named pipe</i> )                                 |
| = | Socket                                                              |
| @ | Symbolischer Link                                                   |
| + | Versteckte Datei (nur auf AIX) oder kontextabhängig (nur auf HP/UX) |
| : | Spezielle Netzwerkdatei (nur auf HP/UX)                             |

Ist die Shellvariable **listlinks** gesetzt, werden symbolische Links genauer klassifiziert:

- @ Symbolischer Link auf eine Datei, die kein Directory ist
- > Symbolischer Link auf ein Directory
- & Symbolischer Link auf Nichts

**listlinks** verlangsamt die Ausführung von **ls-F** und bewirkt, daß bei Dateien, die sich auf anderen Partitionen befinden und auf die symbolische Links verweisen, automatisch die entsprechende Partition montiert wird.

Ist die Shellvariable **listflags** auf **x**, **a**, **A** oder eine beliebige Kombination (wie **xA**) gesetzt, werden diese als Optionen bei **ls-F** verwendet, als ob man **ls -xF**, **ls -Fa**, oder eine Kombination davon (wie **ls -FxA**) aufrufen würde.

**migrate** [-site] *pid* | %*jobid* ... (nur auf TCF)

**migrate** -site (nur auf TCF)

Die erste Aufrufform verlagert den entsprechenden Prozeß oder Job auf die Plattform *site* oder, wenn *site* nicht angegeben, auf die im Systempfad vor-eingestellte Plattform. Die zweite Aufrufform ist äquivalent zu **migrate -site \$\$** (aktuellen Prozeß auf die Plattform *site* verlagern). Ein Verlagern der Shell kann zu Problemen führen.

**newgrp** [-] [*gruppe*]

Mit **newgrp** ist es einem Benutzer möglich, während einer Sitzung in eine andere Gruppe, in der er ebenfalls Mitglied ist, überzuwechseln. Wird **newgrp** ohne die Angabe eines Arguments aufgerufen, wechselt der entsprechende Benutzer in die Gruppe, der er in der Datei */etc/passwd* zugeordnet ist; so kann die Auswirkung eines früher gegebenen **newgrp**-Kommandos wieder rückgängig gemacht werden.

Ist Option - angegeben, werden beim Gruppenwechsel die Shellvariablen so zurückgesetzt, als ob der entsprechende Benutzer sich gerade angemeldet hätte.

**nice** [+*n*] [*kommando*]

**nice** ermöglicht es, Kommandos mit niedrigerer Priorität ausführen zu lassen. Wird **nice** ohne Argumente aufgerufen, wird der *nice*-Wert auf 4 gesetzt. Wird nur +*n* angegeben, wird der *nice*-Wert auf die Zahl *n* gesetzt. Wird **nice** mit einem *kommando* aufgerufen, wird nur dieses *kommando* mit *nice*-Wert 4 bzw. *nice*-Wert *n*, wenn +*n* angegeben ist, ausgeführt. Je größer der *nice*-Wert ist, um so niedriger ist die Priorität. Der Superuser kann für *n* auch einen negativen Wert angeben.

**nohup** [*kommando*]

Wird **nohup** in einem Skript ohne ein Argument aufgerufen, werden für die Dauer der Skript-Ausführung *hangup*-Signale von diesem Skript ignoriert. Wird **nohup** mit einem *kommando* aufgerufen, werden *hangup*-Signale vom *kommando* für die Dauer seiner Ausführung ignoriert.

**notify** [%*job*] ...

bewirkt, daß der Benutzer sofort informiert wird, wenn sich der Status von Jobs ändert. Normalerweise meldet die *tcsh* die Änderung des Status eines Jobs (Beendigung von Hintergrund-Jobs) erst vor der Ausgabe des nächsten Prompts. Werden keine *jobs* beim Aufruf angegeben, bezieht sich **notify** auf den aktuellen Job (wie z.B. zuletzt im Hintergrund gestarteter Job oder angehaltener Job). Für *job* kann eine Nummer, ein String, %, + oder - angegeben werden (siehe auch Jobkontrolle in Kapitel 8.8 und Shellvariable **notify**).

**onintr** [-] [*marke*]

legt die Reaktion der Shell für asynchron eintreffende *intr*-, *hangup*- oder *terminate*-Signale fest. Wird **onintr** ohne jegliche Argumente aufgerufen, wird die default-Signalbehandlung (Shell-Skript beenden) für die entsprechenden Signale eingestellt. **onintr** - bewirkt, daß die entsprechenden Signale ignoriert werden und **onintr** *marke* bewirkt, daß beim Auftreten von Unterbrechungs-Signalen die Marke *marke* angesprungen wird.

Bei Hintergrund-Jobs werden alle Unterbrechungs- und Beendigungs-Signale ignoriert, unabhängig davon, was mit **onintr** als Signalbehandlung eingestellt wurde.

**popd** [-p][-l][-n | -v] [+*n*]

Wird **popd** ohne Argument aufgerufen, wird das an zweitoberster Stelle des Directorystacks stehende Directory zum neuen Working-Directory; der an oberster Stelle des Directorystacks stehende Pfad wird dabei entfernt.

Ist +*n* angegeben, wird der *n*.te Pfad aus dem Directorystack entfernt. Die Stack-Elemente sind dabei von 0 (oberstes Stack-Element) beginnend aufsteigend durchnummeriert.

Bei jedem Aufruf von **popd** wird wie bei **dirs** der Directorystack ausgegeben. Um diese automatische Ausgabe auszuschalten, steht die Shellvariable **pushdsilent** zur Verfügung. Ist die Option **-p** angegeben, wird der Directorystack auch ausgegeben, wenn **pushdsilent** gesetzt ist. Die Optionen **-l**, **-n** und **-v** haben dieselbe Bedeutung wie beim Kommando **dirs**.

**printenv** [*name*]

gibt die Namen und Werte aller Environment-Variablen bzw. nur den Wert der Environment-Variable *name* aus.

**pushd** [-p][-l][-n|-v] [*name* | +*n*]

Mit dem builtin-Kommando **pushd** kann ein neues Directory zum Directorystack hinzugefügt bzw. der Inhalt des Directorystacks rotiert oder einzelne Einträge vertauscht werden.

Wird **pushd** ohne jegliche Argumente aufgerufen, vertauscht es die beiden obersten Directory-Pfade des Stacks.

Ist die Variable **pushdthome** gesetzt, ist ein **pushd**-Aufruf ohne Argumente identisch zu **pushd ~** (wie bei **cd**).

Wird ein *name* angegeben, wird dieser *name* an oberster Stelle des Directorystacks abgelegt und somit das neue Working-Directory; der Pfadname des alten Working-Directorys befindet sich dann an zweitoberster Stelle des Directorystacks.

Wird anstelle von *name* ein Bindestrich (-) angegeben, wird das vorherige Working-Directory an oberster Stelle des Directorystacks abgelegt und somit das neue Working-Directory; der Pfadname des alten Working-Directorys befindet sich dann an zweitoberster Stelle des Directorystacks.

Ist die Variable **dunique** gesetzt, entfernt **pushd** alle Vorkommen von *name* aus dem Directorystack, bevor es die entsprechenden Aktionen durchführt: *name* wird Working-Directory und an oberster Stelle des Stacks gespeichert.

Ist +*n* angegeben, wird der Directorystack so rotiert, daß das *n*. te Element (Pfadname) das oberste Stackelement wird. Die Stackelemente sind dabei von 0 (oberstes Stackelement) beginnend aufsteigend durchnummeriert.

Ist jedoch die Variable **dextract** gesetzt, entfernt **pushd** +*n* das *n*. te Element vom Directorystack und speichert es an oberster Stelle des Directorystacks, was einem Wechsel zu diesem Directory bewirkt.

Bei jedem Aufruf von **pushd** wird wie bei **dirs** der Directorystack ausgegeben. Um diese automatische Ausgabe auszuschalten, steht die Shellvariable **pushdsilent** zur Verfügung. Ist die Option **-p** angegeben, wird der Directorystack auch ausgegeben, wenn **pushdsilent** gesetzt ist. Die Optionen **-l**, **-n** und **-v** haben dieselbe Bedeutung wie beim Kommando **dirs**.

```
> dirs
~ /usr/lib /usr/bin /etc
> set dextract
> pushd +2 [nur 2. (eigentliches 3.) Element nach vorne ziehen]
/usr/bin ~ /usr/lib /etc
> unset dextract
> pushd +2 [nun wird Directorystack ab 2. (eigentliches 3.) Element
 rotiert]
/usr/lib /etc /usr/bin ~
```

**rehash**

bewirkt, daß die interne Hashing-Tabelle zunächst gelöscht und dann wieder neu aufgebaut wird. **rehash** muß immer dann aufgerufen werden, wenn während einer Sitzung neue Programme in den **path**-Directories angelegt wurden. Da die Hashing-Tabelle bereits vorher aufgebaut wurde, würden diese Programme niemals in der veralteten Hashing-Tabelle gefunden, außer sie werden mit dem absoluten Pfadnamen aufgerufen.

### **repeat** *n kommando*

Das angegebene *kommando*, das sich in der gleichen Zeile wie **repeat** befinden muß, wird *n* mal ausgeführt. Eventuell angegebene Ein-/Ausgabeumlenkungen werden dabei nur einmal vorgenommen und zwar sogar dann, wenn für *n* der Wert **0** angegeben wurde. Für *kommando* muß ein einfaches Kommando angegeben sein. Es darf also kein Alias, keine Pipeline, Kommandoliste oder geklammerte Kommandoliste dafür angegeben werden.

### **rootnode** *//knotenname* (nur auf Domain / OS)

ändert den root-Knoten auf *//knotenname*, so daß die Angabe von */* als *//knotenname* interpretiert wird.

### **sched** (1)

### **sched** *[+]hh:mm kommando* (2)

### **sched** *-n* (3)

Mit dem builtin-Kommando **sched** können Kommandos festgelegt werden, die von der tcsh zu einer bestimmten angegebenen Zeit ablaufen sollen.

- (1) Wird **sched** ohne jegliche Argumente aufgerufen, werden alle momentan mit **sched** eingerichteten Kommandos mit der Zeit, an der sie auszuführen sind, ausgegeben. Über die Shellvariable **sched** kann das Format einer solcher Ausgabe festgelegt werden.
- (2) Mit der zweiten Aufrufform kann ein neues Kommando eingerichtet werden, das zu einer bestimmten Zeit auszuführen ist, wie z.B.

### **sched 11:55 echo "Es ist Zeit fuer das Mittagessen"**

Durch diese Eingabe wird die tcsh angewiesen, um 11:55 den Text *Es ist Zeit fuer das Mittagessen* auszugeben. Die Zeit kann dabei im 24-Stunden-Format (wie z.B. 18:40) oder im 12-Stunden-AM/PM-Format (wie z.B. 6:40pm) festgelegt werden, wie z.B.

### **sched 5pm set prompt='[%h] 5 Uhr; Zeit nach hause zu gehen'**

Auch kann die Zeit relativ zur aktuellen Zeit angegeben werden, wie z.B.:

### **sched +1:30 mailx hans <termin**

### **sched +2am echo "wieder 2 Stunden vorbei" [nicht erlaubt]**

Bei einer relativen Zeitangabe darf nicht das AM / PM-Format verwendet werden.

- (3) Mit der dritten Aufrufform kann ein mit **sched** eingerichtetes Kommando wieder entfernt werden, wie z.B.

```
> sched
 1 Fri Jan 26 13:35 mailx hans
 2 Fri Jan 26 17:00 set prompt=[%h] 5 Uhr; Zeit nach hause zu gehen
> sched -2
> sched
 1 Fri Jan 26 13:35 mailx hans
```

**sched** ist dem Kommando **at** sehr ähnlich. **sched** hat aber gegenüber **at** den Nachteil, daß es die Kommandos eventuell nicht genau zur festgelegten Zeit ausführt, z.B. wenn gerade zu diesem Zeitpunkt ein anderes Programm in der tcsh abläuft. In diesem Fall wird das entsprechende **sched**-Kommando erst nach der Beendigung des gerade ablaufenden Kommandos ausgeführt. Ein Vorteil von **sched** gegenüber **at** ist, daß das builtin-Kommando **sched** direkt von der tcsh ausgeführt wird und somit Zugriff auf interne Shell-Konstrukte wie Aliase, Shellvariable usw. hat. Mit **sched** ist es z.B. auch möglich, zu einer bestimmten Zeit das Working-Directory zu wechseln, was mit **at** nicht möglich ist.

|                                |     |
|--------------------------------|-----|
| <b>set</b>                     | (1) |
| <b>set name ...</b>            | (2) |
| <b>set name=wert ...</b>       | (3) |
| <b>set name=(wortliste)</b>    | (4) |
| <b>set name[index]=wert</b>    | (5) |
| <b>set -r</b>                  | (6) |
| <b>set -r name ...</b>         | (7) |
| <b>set -r name=wert ...</b>    | (8) |
| <b>set -r name=(wortliste)</b> | (9) |

**set** ermöglicht das Setzen / Anzeigen von Variablen bzw. das Zuweisen von Werten / Wortlisten / Attributen an Variablen:

- (1) Gibt alle momentan definierten Shellvariablen mit ihren Werten aus.
- (2) Weist der Variablen *name* den Nullstring zu.
- (3) Weist der Variablen *name* den Wert *wert* zu.
- (4) Weist dem Array *name* die Werte von *wortliste* zu; dabei wird das erste Wort *name[1]*, das zweite *name[2]* usw. zugewiesen.
- (5) Weist dem Array-Element *name[index]* den Wert *wert* zu; dabei muß *name[index]* bereits existieren.

Bei den Aufrufformen (3) bis (5) wird immer Kommando-, Variablen- und Dateinamen-Substitution für den rechten Teil der Zuweisung durchgeführt.

(6) Gibt alle auf *readonly* festgelegten Shellvariablen mit ihren Werten aus.

(7) Setzt Variable *name* auf *readonly*, auch wenn sie noch keinen Wert hat.

(8) und (9) entsprechen (3) und (4), nur daß nach der Zuweisung die Variable *name* auf *readonly* festgelegt wird.

Auch ist es möglich, mehrere Aufrufformen in einem **set**-Kommando zu verwenden, wie z.B.

**set a=(brief\*) b=15 x[3]=7** bzw.

**set -r a fuenf = 5 dual=( 0 1)**

Variablen-Substitution findet immer zuerst für alle Argumente der **set**-Kommandozeile statt, bevor die eigentlichen Zuweisungen erfolgen.

Links und rechts vom = können Leer- und Tabzeichen angegeben sein. Wenn aber welche angegeben sind, dann müssen sie sowohl links wie auch rechts vom = angegeben werden.

**setenv** [*name* [*wert*]]

**setenv** definiert eine Environment-Variable *name* mit dem angegebenen *wert*. Shellvariablen, die mit **set** definiert werden, werden normalerweise nicht in die Environment von Kindprozessen exportiert. Die Ausnahmen dabei sind die Shellvariablen **term** und **path**. Mit **setenv** definierte Variablen werden dagegen exportiert.

Ist kein *wert* angegeben, wird die Environment-Variable *name* mit dem Leerstring besetzt.

Wird **setenv** ohne jegliche Argumente aufgerufen, gibt es alle momentan definierten Environment-Variablen mit ihren Werten aus.

**setpath** *pfad* (nur auf Mach)

siehe auch Man-page zu **setpath**.

**setspath** **LOCAL** | *site* | *cpu* ... (nur auf TCF)

setzt den System-Ausführungspfad.

**settc** *cap wert*

legt für die Shell fest, daß die Terminal-capability *cap* den angegebenen *wert* hat.

**setty** [-**d** | -**q** | -**x**] [-**a**] [[+ | -]*flag*] ...

Die tcsh kennt drei verschiedene Terminal-Modi: *edit* (für das Editieren von Kommandozeilen), *quote* (Für das Ausschalten der Bedeutung von Sonderzeichen) und *execute* (für das Ausführen von Kommandos). Einige Einstellungen bleiben dabei im jeweiligen Terminal-Modus immer konstant, so daß



Kommandos, die den Terminal in einem korruptierten Zustand hinterlassen, keine Auswirkung auf die aktuelle Shell haben.

Mit **setty** kann nun zunächst der entsprechende Terminal-Modus eingestellt werden: **-d** (*edit*), **-q** (*quote*) oder **-x** (*execute*). Wird keine dieser drei Optionen angegeben, wird die *execute*-Einstellung verwendet. Wird **setty** ohne *flag*-Argumente aufgerufen, gibt es die konstanten Einstellungen für den über Optionen ausgewählten Terminal-Modus bzw. den default-Modus (*execute*) aus, wie z.B.:

```
> setty
iflag:-inlcr -igncr +icrnl
oflag:+opost +onlcr -onlret
cflag:
lflag:+isig +icanon +echo +echoe -echonl -noflsh +echoctl -flusho +iexten
chars:
> setty -q
iflag:-ixon -ixoff
oflag:
cflag:
lflag:+isig -iexten
chars:
```

Bei dieser Ausgabe zeigt ein vorangestelltes Plus (+) an, daß diese Flags immer eingeschaltet bleiben, und ein vorangestelltes Minus (-) zeigt an, daß diese Flags immer ausgeschaltet bleiben.

Ist die Option **-a** angegeben, werden alle Einstellungen des entsprechend ausgewählten Modus ausgegeben, unabhängig davon, ob sie konstant sind oder nicht.

Eventuelle *flag*-Angaben haben die folgende Auswirkung:

**+flag** *flag* bleibt ab jetzt immer eingeschaltet

**-flag** *flag* bleibt ab jetzt immer ausgeschaltet

**flag** *flag* ist ab jetzt nicht mehr konstant und kann ein- oder ausgeschaltet werden

So wird z.B. mit folgendem Aufruf

```
setty -x +echok echoe [-x könnte auch weggelassen werden]
```

festgelegt, daß bei der Ausführung von Kommandos immer das Flag *echok* eingeschaltet wird, während das Flag *echoe* von den Kommandos beliebig ein- oder ausgeschaltet werden kann.

**setxVers** [*string*] (nur auf TCF)

setzt das experimentelle Versions-Präfix auf *string* bzw. löscht das momentane Versions-Präfix, wenn *string* nicht angegeben ist.

**shift** [*variable*]

Wird **shift** ohne ein Argument aufgerufen, werden die Elemente von **argv** um eine Position nach links geschoben, so daß der Wert **\$argv[2]** dem Positionsparameter **argv[1]**, der Wert **\$argv[3]** dem Positionsparameter **argv[2]** usw. zugewiesen wird.

Wird eine *variable* angegeben, werden die einzelnen Werte dieser *variable* um eine Position nach links geschoben.

Wenn **argv** bzw. die angegebene *variable* keinen Wert mehr besitzt, wertet **shift** dies als Fehler.

**source** [-h] *dateiname* [*argument(e)*]

ist das Gegenstück der **tcsh** zum Punkt-Kommando der Bourne-Shell. **source** liest die Kommandos der angegebenen Datei *dateiname* und führt sie in der aktuellen **tcsh** aus. Die dabei gelesenen Kommandos werden nicht in die History-Liste mit aufgenommen. **source**-Aufrufe dürfen geschachtelt werden, d.h., daß sich in der Datei *dateiname* wiederum **source**-Aufrufe befinden dürfen.

Sind *argument(e)* angegeben, werden diese in **argv** abgelegt. Ist die Option **-h** angegeben, werden die gelesenen Kommandos nicht ausgeführt, sondern in der History-Liste mit aufgenommen, was weitgehend dem Aufruf **history -L** entspricht.

**stop** %*job* | *pid*...

**stop** bewirkt, daß die Ausführung der angegebenen Hintergrund-*jobs* und Prozesse angehalten wird. Für *job* kann eine Nummer, ein String, %, + oder - angegeben werden (siehe auch Jobkontrolle in Kapitel 8.8).

**suspend**

hält die aktuelle **tcsh** an; **suspend** wird meist verwendet, wenn am gleichen Terminal mit dem Kommando **su** (**switch user**) auf eine andere Benutzerkennung ohne Verlassen der aktuellen **tcsh** umgeschaltet wurde. Mit **suspend** kann dann diese neue Sitzung kurzzeitig unterbrochen werden, um zwischenzeitlich in der aktuellen **tcsh** zu arbeiten.

**switch** (*wort*)

**case** *pattern*<sub>1</sub>:

*kdos*<sub>1</sub>

**breaksw**

**case** *pattern*<sub>2</sub>:

*kdos*<sub>2</sub>

**breaksw**

:

[ **default**:

*kdos<sub>n</sub>*  
**breaksw**]

### **endsw**

Die **tcsh** führt zuerst für das angegebene *wort* Variablen-, Kommando- und Dateinamen-Substitution durch. Den daraus resultierenden String vergleicht sie dann der Reihe nach mit den bei **case** angegebenen *pattern*. Findet sie ein übereinstimmendes *pattern*, führt sie die nachfolgenden *kdos* aus, bis sie **breaksw** liest; **breaksw** veranlaßt das Verlassen der **switch**-Anweisung. Wird eine zu einer **case**-Marke angegebene Kommandofolge nicht mit einem **breaksw** beendet, wird die Ausführung mit der Kommandofolge der nächsten **case**-Marke fortgesetzt.

Paßt keines der angegebenen *pattern*, werden, wenn die **default**-Marke vorhanden ist, die danach angegebenen Kommandos ausgeführt, oder die **switch**-Anweisung ohne jegliche weitere Aktion verlassen.

In den *pattern* können die Metazeichen \*, ? und [...] der Dateinamenexpansion verwendet werden.

**case**-Marken und **default** müssen am Anfang einer Zeile angegeben sein.

### **telltc**

Gibt die Werte aller Terminal-*capabilities* aus.

### **time** [*kommando*]

Ist ein *kommando* angegeben, gibt **time** nach der Ausführung des Kommandos die von diesem *kommando* verbrauchte Zeit entsprechend den Vorgaben in der Shellvariablen **time** aus. Für *kommando* muß ein einfaches Kommando angegeben sein. Es darf also kein Alias, keine Pipeline, Kommandoliste oder geklammerte Kommandoliste dafür angegeben werden. Wird **time** ohne ein Argument aufgerufen, gibt es die von der aktuellen tcsh und allen ihren Kindprozessen bis zum jetzigen Zeitpunkt verbrauchte Zeit aus.

### **umask** [3-stellige-oktaltzahl]

setzt die Dateikreierungsmaske auf den Wert 3-stellige-oktaltzahl. Die Dateikreierungsmaske legt mittels ihres Bitmusters fest, welche Zugriffsrechte beim Anlegen neuer Dateien oder Directories immer zu entfernen sind. Wird **umask** ohne die Angabe eines Arguments aufgerufen, gibt **umask** lediglich den Wert der momentanen Dateikreierungsmaske aus.

### **unalias** *pattern*

löscht alle Aliase, deren Namen durch das angegebene *pattern* abgedeckt werden. In *pattern* sind dabei die Metazeichen \*, ? und [...] der Dateinamenexpansion erlaubt; so würde z.B. der Aufruf **unalias \*** alle momentan definierten Aliase löschen. Falls das angegebene *pattern* keinen einzigen Aliasnamen abdeckt, wird dies nicht als Fehler gewertet.

**uncomplete pattern**

löscht alle vom Benutzer (mit **complete**) definierten Vervollständigungen, deren Namen durch das angegebene *pattern* abgedeckt werden. In *pattern* sind dabei die Metazeichen **\***, **?** und **[..]** der Dateinamenexpandierung erlaubt; so würde z.B. der Aufruf **uncomplete \*** alle momentan definierten Vervollständigungen löschen. Falls das angegebene *pattern* keine einzige Vervollständigung abdeckt, wird dies nicht als Fehler gewertet.

**unhash**

schaltet die Verwendung der internen Hashing-Tabelle durch die *tcsh* aus.

**unlimit [-h] [resource]**

hebt das für *resource* festgelegte Limit wieder auf. Ist *resource* nicht angegeben, werden die Limits für alle System-Ressourcen entfernt. Ist die Option **-h** angegeben, was nur dem Superuser erlaubt ist, werden die Limits für die entsprechenden Ressourcen aufgehoben (siehe auch builtin-Kommando **limit**).

**unset pattern**

löscht alle Shellvariablen, deren Namen durch das angegebene *pattern* abgedeckt werden, wenn diese nicht als *readonly* markiert wurden. In *pattern* sind dabei die Metazeichen **\***, **?** und **[..]** der Dateinamenexpandierung erlaubt. So würde z.B. **unset \*** alle momentan definierten Shellvariablen löschen, die nicht mit *readonly* markiert sind. Falls das angegebene *pattern* keine einzige Shellvariable abdeckt, wird dies nicht als Fehler gewertet.

**unsetenv pattern**

löscht alle Environment-Variablen, deren Namen durch das angegebene *pattern* abgedeckt werden. In *pattern* sind dabei die Metazeichen **\***, **?** und **[..]** der Dateinamenexpandierung erlaubt; so würde z.B. der Aufruf **unsetenv \*** alle momentan definierten Environment-Variablen löschen. Falls das angegebene *pattern* keine einzige Environment-Variable abdeckt, wird dies nicht als Fehler gewertet.

**ver [systype [kommando]]** (nur auf Domain / OS)

Ohne Argumente gibt **ver** den Inhalt der Environment-Variable **SYSTYPE** aus. Ist *systype* angegeben, wird die Environment-Variable **SYSTYPE** auf *systype* gesetzt. Ist *systype* und ein *kommando* angegeben, wird das *kommando* unter dem System *systype* ausgeführt. Für *systype* kann *bsd4.3* oder *sys5.3* angegeben werden.

**wait**

Veranlaßt die *tcsh* auf die Beendigung aller Hintergrund-Prozesse zu warten.

**watchlog**

ein anderer Name für das builtin-Kommando **log**. **watchlog** ist nur verfügbar, wenn die tcsh entsprechend kompiliert wurde (siehe auch Shellvariable **version**).

**where** *kdo\_name*

Gibt alle Vorkommen vom angegebenen Kommando *kdo\_name* aus: Alias, Builtin-Kommando, ausführbare Programme gefunden über die **path**-Variable.

**which** *kdo\_name*

Gibt aus, welches Programm ausgeführt wird, wenn der angegebene *kdo\_name* aufgerufen wird (Alias, builtin-Kommando, Pfad eines Programms usw.). Dieses builtin-Kommando entspricht weitgehend dem Kommando **which** (siehe auch Man-page), nur daß das builtin-Kommando 10- bis 100mal schneller abläuft und spezielle tcsh-Aliase und -Builtin-Kommandos erkennt, was das andere **which**-Kommando nicht kann (siehe auch Editor-Kommando *which-command*, das auf **M-?** voreingestellt ist).

**while** (*ausdruck*)*kdos***end**

Solange die Auswertung von *ausdruck* TRUE liefert, werden die *kdos* in der **while**-Schleife ausgeführt. Die Schlüsselwörter **while** und **end** müssen als erstes Wort einer Zeile angegeben sein.

Wird die **while**-Schleife interaktiv eingegeben, zeigt die tcsh, solange nicht das abschließende **end** eingegeben wurde, mit der Ausgabe des Prompts *while?* oder des Inhalts von **prompt2** an, daß die **while**-Schleife noch nicht abgeschlossen ist.

## 8.12 Spezielle Aliase

Ist eines der nachfolgenden Aliase gesetzt, wird es zum geeigneten Zeitpunkt automatisch ausgeführt. Die Voreinstellung ist, daß die Aliase nicht gesetzt sind.

**beepcmd** Wird ausgeführt, wenn die tcsh die Terminalglocke läuten will.

**cwddcmd** Wird bei jedem Wechsel des Working-Directorys ausgeführt. So bewirkt z.B. die Festlegung

```
alias cwddcmd 'echo -n "--- $cwd --- "; date'
```

daß bei jedem Directory-Wechsel das neue Working-Directory gefolgt von der aktuellen Zeit und dem aktuellen Datum ausgegeben wird. In **cwddcmd** sollte keines der Kommandos **cd**, **pushd**

- oder **popd** aufgerufen werden, da dies zu einer Endlosschleife führen kann.
- periodic** Wird alle **tperiod**-Minuten ausgeführt. Wenn z.B. folgendes festgelegt wurde,
- ```
> set tperiod = 60
> alias periodic 'echo - Wieder ne Stunde vorbei -;date'
```
- wird jede Stunde der Text - *Wieder ne Stunde vorbei* - gefolgt von der aktuellen Zeit und dem aktuellen Datum ausgegeben. Ist **periodic** gesetzt, aber **tperiod** entweder auf 0 oder überhaupt nicht gesetzt, verhält sich **periodic** genauso wie **precmd**.
- precmd** Wird vor der Ausgabe jedes Prompts ausgeführt. So bewirkt z.B. die Festlegung
- alias precmd date**
- daß vor jeder Ausgabe des Prompts das aktuelle Datum angezeigt wird.
- shell** Legt den Interpreter für ausführbare Skripts fest, die nicht selbst in der ersten Zeile mit *#!interpreter* ihren eigenen Interpreter auswählen. Das erste Wort in **shell** sollte ein voller Pfadname sein, wie z.B. */bin/csh* oder */usr/bin/tcsh*.

8.13 Spezielle Shellvariablen

Die nachfolgenden Shellvariablen haben eine spezielle Bedeutung für die Shell. Dabei ist zu unterscheiden zwischen:

Variablen, die automatisch beim tcsh-Start gesetzt werden

addsuffix, **argv**, **autologout**, **command**, **echo_style**, **edit**, **gid**, **group**, **home**, **loginsh**, **oid**, **path**, **prompt**, **prompt2**, **prompt3**, **shell**, **shlvl**, **tcsh**, **term**, **tty**, **uid**, **user** und **version**. Die dabei zugewiesenen Werte behalten diese Variablen solange, bis sie explizit vom Benutzer in der jeweiligen Sitzung geändert werden.

Variablen, die ständig von tcsh aktualisiert werden

cwd, **dirstack**, **owd**, **status** und **logout** (beim Abmelden).

Variablen, die mit Environment-Variablen synchronisiert werden

Zu den Variablen **afsuser**, **group**, **home**, **path**, **shlvl**, **term** und **user** existieren gleichnamige Environment-Variablen (in Großschreibung). Immer wenn die gleichnamige Environment-Variable geändert wird, wird diese Ände-

rung auch für die entsprechende Variable durchgeführt, wenn diese nicht mit *readonly* markiert ist. Umgekehrt wirkt sich eine Änderung einer der obigen Shellvariablen auch auf die gleichnamige Environment-Variable aus. Es ist darauf hinzuweisen, daß die beiden Variablen **cwd** und **PWD** zwar die gleiche Bedeutung haben, aber nicht wie die obigen synchronisiert sind. Die Anpassung der unterschiedlichen Formate zwischen **PATH** (Doppelpunkt ist Trennzeichen) und **path** (Leerzeichen ist Trennzeichen) wird von der tcsh automatisch durchgeführt.

addsuffix

Wenn gesetzt, wird bei Dateinamen-Vervollständigung automatisch am Ende eines Directorynamens ein Slash / und am Ende des Namens einer normalen Datei ein Leerzeichen angehängt. Voreinstellung: gesetzt.

afsuser

Wenn gesetzt, verwendet der **autologout**-Mechanismus für das Sperren des Bildschirms (*autolock*) diesen Wert anstelle des lokalen Benutzernamens zur Authentifizierung (*authentication*).

ampm

Wenn gesetzt, werden alle Zeiten im 12-Stunden AM/PM-Format ausgegeben.

argv

Argumente der Shell. **argv[1]** entspricht **\$1**, **argv[2]** entspricht **\$2** usw. In interaktiven Shells meist leer.

autocorrect

Wenn gesetzt, wird automatisch das Editor-Kommando *spell-word* vor jedem Vervollständigungsversuch aufgerufen.

autoexpand

Wenn gesetzt, wird automatisch das Editor-Kommando *expand-history* vor jedem Vervollständigungsversuch aufgerufen.

autolist

Wenn gesetzt, werden bei einem Wort, das nicht vollkommen vervollständigt werden kann, alle Vervollständigungsmöglichkeiten ausgegeben. Wird die Variable **autolist** auf *ambiguous* gesetzt, werden die Vervollständigungsmöglichkeiten nur dann angezeigt, wenn die Vervollständigung mehrere Möglichkeiten zuläßt und keine neuen Zeichen dem Wort hinzugefügt wurden.

autologout

Wenn gesetzt, beendet sich die tcsh automatisch, wenn die als erstes Wort angegebenen Minuten verstrichen sind, ohne daß irgendwelche Eingaben erfolgten. Das optionale zweite Wort legt die Minuten fest, nach denen der Bildschirm zu sperren ist, wenn in einer solchen Zeitspanne keine Eingaben getätigt werden.

Wenn die tcsh sich automatisch beendet, gibt sie *auto-logout* aus und setzt die Variable **logout** auf *automatic*.

Wenn die tcsh sich automatisch sperrt, muß der Benutzer zuerst sein Paßwort eingeben, bevor er weiterarbeiten kann. Fünf Fehlversuche resultieren dabei dann ebenfalls in einer automatischen Beendigung der tcsh.

Voreinstellung ist **autologout = 60** (automatisches Abmelden nach 60 Minuten und kein automatisches Sperren) bei Login- und Superuser-Shells. Diese Voreinstellung gilt jedoch nicht, wenn die tcsh unter X Windows läuft (Environment-Variable **DISPLAY** ist gesetzt), der Terminal ein Pseudo-Terminal ist oder die tcsh nicht entsprechend kompiliert wurde (siehe auch Variable **version**).

backslash_quote

Wenn gesetzt, kann mit Backslash (\) die Bedeutung der Sonderzeichen \, ".." und '.' ausgeschaltet werden. Dies kann Syntax-Fehler in csh-Skripts nach sich ziehen.

cdpath

Enthält eine Liste von Directories, in denen das Kommando **cd** suchen soll, wenn es das beim Aufruf angegebene Directory nicht im aktuellen Directory findet.

command

Wenn gesetzt, enthält **command** das Kommando, das der tcsh mit der Option **-c** übergeben wurde.

complete

Ist **complete** auf *enhance* gesetzt, gilt bei Vervollständigungen folgendes:

- ▶ Es wird nicht zwischen Groß- und Kleinschreibung unterschieden.
- ▶ Punkt (.), Querstrich (-) und Unterstrich (_) gelten als Wort-Trennzeichen.
- ▶ Querstrich (-) und Unterstrich (_) werden nicht unterschieden.

correct

Über diese Variable ist es möglich festzulegen, was zu korrigieren ist. Hat **correct** den Wert *cmd*, wird nur der Kommandoname bei der Übergabe der Kommandozeile mit *Return* korrigiert. Ist **correct** auf *all* gesetzt, wird bei *Return* die ganze Kommandozeile korrigiert. Ist **correct** auf *complete* gesetzt, wird bei *Return* der Kommandoname automatisch vervollständigt.

cwd

enthält den vollen Pfadnamen des Working-Directorys.

dextract

Wenn gesetzt, rotiert **pushd +n** nicht den Directorystack, sondern entfernt das *n*. te Element vom Directorystack und speichert es an oberster Stelle des Directorystacks, was einen Wechsel zu diesem Directory bewirkt.

dirsfile

Legt die Datei fest, in der **dirs -S** bzw. **dirs -L** die History-Liste speichert (**-S** bzw. liest **-L**), wenn bei diesen Aufrufen kein Dateiname angegeben ist. Ist **dirsfile** nicht gesetzt, wird *~/cshdirs* verwendet. Da normalerweise beim Anmelden nur *~/tcshrc* vor *~/cshdirs* gelesen wird, sollte **dirsfile** in *~/tcshrc* und nicht in *~/login* gesetzt werden.

dirstack

Enthält immer die einzelnen Einträge des Directorystacks (**\$dirstack[1]**, **\$dirstack[2]**, ...). Es ist zu beachten, daß **\$dirstack[1]** und **=0** (nicht **=1**) immer das Working-Directory ist. Ein Verändern der einzelnen Worte von **dirstack** hat dabei auch die entsprechende Auswirkung auf den Directorystack, wobei jedoch **\$dirstack[1]** immer das Working-Directory bleibt.

dunique

Wenn gesetzt, entfernt ein **pushd name** zunächst alle Vorkommen von *name* aus dem Directorystack, bevor es die entsprechenden Aktionen durchführt: *name* wird Working-Directory und an oberster Stelle des Directorystacks gespeichert.

echo

Wenn gesetzt, dann wird jede Kommandozeile vor ihrer Ausführung nochmals angezeigt. Bei nicht builtin-Kommandos haben dabei bereits alle Substitutionen und Expandierungen stattgefunden; bei builtin-Kommandos dagegen werden Kommando- und Dateinamen-Substitution erst danach durchgeführt. **echo** wird automatisch bei Angabe der tcsh-Option **-x** gesetzt.

echo_style

Legt das Verhalten des builtin-Kommandos **echo** fest. **echo_style** kann auf folgende Werte gesetzt werden:

bsd keine Ausgabe eines Neuezeile-Zeichens, wenn das erste Argument **n** ist.

sysv Escape-Sequenzen im **echo**-Text werden erkannt

both *bsd* und *sysv* zusammen; Voreinstellung

none weder *bsd* noch *sysv*

edit

Wenn gesetzt, wird der Kommandozeilen-Editor eingeschaltet. Bei interaktiven Shells wird **edit** automatisch gesetzt.

ellipsis

Wenn gesetzt, dann werden bei Ausgabe des Prompts (Formate **%c**, **%.** und **%C**) nicht angezeigte Directories mit drei Punkten ... (Ellipse) angezeigt (siehe auch Variable **prompt**).

ignore

Die in **ignore** enthaltenen Suffizes werden bei Wortvervollständigungen ignoriert.

filec

In der tcsh ist immer die automatische Vervollständigung eingeschaltet. Ist **filec** in der C-Shell gesetzt, wird die Dateinamenvervollständigung dort eingeschaltet.

gid

Enthält die reale group-ID des Benutzers.

group

Enthält den Gruppennamen des Benutzers.

histchars

Definiert die History-Metazeichen. Das erste Zeichen ersetzt **!** und das zweite Zeichen ersetzt **^**. Die Voreinstellung ist: **histchars="!^"**.

histdup

Legt fest, wie mit History-Zeilen zu verfahren ist, die identisch sind. **histdup** kann auf die folgenden Werte gesetzt werden:

- all* Eine Kommandozeile wird nur dann in die History-Liste aufgenommen, wenn nicht bereits eine solche Kommandozeile in der History-Liste existiert.
- prev* Eine Kommandozeile wird nur dann in die History-Liste aufgenommen, wenn sie sich von der letzten Kommandozeile in der History-Liste unterscheidet.
- erase* Wenn in der History-Liste bereits eine Kommandozeile existiert, die gleich der aktuellen Kommandozeile ist, wird die alte Kommandozeile aus der History-Liste entfernt und dafür die neue Kommandozeile in die History-Liste aufgenommen.

histfile

Legt die Datei fest, in der **history -S** bzw. **history -L** die History-Liste speichert (-S) bzw. liest (-L), wenn bei diesen Aufrufen kein Dateiname angegeben ist. Ist **histfile** nicht gesetzt, wird *~/history* verwendet. **histfile** ist hilfreich, wenn man das gleiche Home-Directory zwischen verschiedenen Maschinen verwendet, oder wenn man unterschiedliche History-Listen für verschiedene Terminals unterhalten möchte. Da normalerweise beim Anmelden nur *~/tcshrc* vor *~/history* gelesen wird, sollte **histfile** in *~/tcshrc* und nicht in *~/login* gesetzt werden.

histlit

Die *tcsh* hebt die eingegebenen Kommandozeilen immer in zwei verschiedenen Darstellungsformen in der History-Liste auf: mit und ohne expandierte History-Substitutionen. Um zwischen beiden Darstellungsformen hin- und herschalten zu können, steht das Editor-Kommando *toggle-literal-history* (voreingestellt auf **M-R**) zur Verfügung. Ist **histlit** gesetzt, ist die nicht expandierte Form eingestellt.

history

Das erste Wort dieser Variablen legt fest, wie viele der zuletzt eingegebenen Kommandos in der History-Liste aufzuheben sind. Das optionale zweite Wort legt das Format fest, in dem die History-Liste auszugeben ist. Ist dieses zweite Wort nicht angegeben, wird das Format **%h\t%T\t%R\n** verwendet (siehe auch Variable **prompt**). Voreinstellung ist: **history = 100**.

home

Home-Directory des Benutzers. Ein *~* (Tilde) bei Dateinamenexpansion greift auf diese Variable zu.

ignoreeof

Wenn gesetzt, legt ihr Wert die Anzahl der EOFs (*Strg-D*) fest, die unmittelbar nacheinander einzugeben sind, bevor die tcsh sich beendet.

Ist **ignoreeof** auf 0 oder den Leerstring gesetzt, kann die tcsh niemals mit EOF (*Strg-D*) beendet werden. Versucht der Benutzer in diesem Fall die tcsh mit EOF zu beenden, meldet sie lediglich *Use "exit" to leave tcsh*.

Ist **ignoreeof** nicht gesetzt, wird 1 als Wert von **ignoreeof** angenommen.

inputmode

Wenn auf *insert* gesetzt, wird der Kommandozeilen-Editor am Anfang einer Zeilen-Eingabe auf den Einfüge-Modus gesetzt. Ist **inputmode** auf *overwrite* gesetzt, wird der Kommandozeilen-Editor am Anfang einer Zeilen-Eingabe auf den Überschreibe-Modus gesetzt.

listflags

Ist **listflags** auf **x**, **a**, **A** oder eine beliebige Kombination (wie **xA**) gesetzt, werden diese als Optionen bei **ls-F** verwendet, als ob man **ls -xF**, **ls -Fa**, **ls -FA** oder eine Kombination davon (wie **ls -FxA**) aufrufen würde.

- a** Listet alle Dateien, auch die, die mit Punkt (.) beginnen.
- A** Listet alle Dateien, auch die, die mit Punkt (.) beginnen, außer . (Punkt) und .. (Punkt-Punkt).
- x** Sortiert die Dateien bei der Ausgabe von links nach rechts und nicht von oben nach unten.

Ist in **listflags** ein zweites Wort angegeben, legt es den Pfad des aufzurufenden **ls**-Kommandos fest.

listjobs

Wenn gesetzt, werden bei **^Z** (*suspend*) immer alle momentanen Jobs aufgelistet, wie beim Aufruf des builtin-Kommandos **jobs**. Ist **listjobs** auf *long* gesetzt, werden dabei die Jobs wie bei **jobs -l** aufgelistet.

istlinks

Wenn gesetzt, werden symbolische Links genauer klassifiziert:

- @ Symbolischer Link auf eine Datei, die kein Directory ist
- > Symbolischer Link auf ein Directory
- & Symbolischer Link auf Nichts

listlinks verlangsamt die Ausführung von **ls-F** und bewirkt, daß bei Dateien, die sich auf anderen Partitionen befinden und auf die symbolische Links verweisen, automatisch die entsprechende Partition montiert wird.

listmax

Legt ein Limit von Namen für die Vervollständigungsmöglichkeiten fest, ab dem der Benutzer beim Editor-Kommando *list-choices* gefragt wird, ob diese wirklich aufzulisten sind.

listmaxrows

Legt ein Limit von Zeilen für die Vervollständigungsmöglichkeiten fest, ab dem der Benutzer beim Editor-Kommando *list-choices* gefragt wird, ob diese wirklich aufzulisten sind.

loginsh

Wird von tcsh gesetzt, wenn sie eine Login-Shell ist. Ein Verändern dieser Variable durch den Benutzer hat keinerlei Auswirkung.

logout

Wird beim Beenden der tcsh gesetzt: auf *normal* bei einer normalen Abmeldung, auf *automatic* bei einer automatischen Abmeldung und auf *hangup*, wenn die tcsh durch ein hangup-Signal abgebrochen wird.

mail

Enthält Pfadnamen oder Directories von mailbox-Dateien. Die Namen sind dabei durch Leer- oder Tabzeichen getrennt. Wenn das erste Wort von **mail** (**\$mail[1]**) eine Zahl ist, legt diese das Zeitintervall für die Überprüfung auf das Ankommen neuer mail (in Sekunden) fest. Voreinstellung für die Überprüfung ist 600 (alle 10 Minuten).

Falls in **mail** ein Directory angegeben ist, betrachtet die tcsh jede in diesem Directory vorhandene Datei als eigene *mail*, und meldet das Ankommen neuer *mail* mit »*You have n mails*« oder »*You have n mails in name*«. Diese Funktionalität ist hauptsächlich für Mail-Systeme wie das *Andrew Mail System* vorgesehen.

Voreinstellung für **mail**: Falls **mail** nicht gesetzt ist, legt die Variable **MAIL** aus der Standardumgebung den Pfadnamen der mailbox-Datei fest.

matchbeep

Kann mit verschiedenen Werten gesetzt werden, die festlegen, ob in bestimmten Situationen von Vervollständigungen der Piepston auszugeben ist oder nicht:

<i>never</i>	Bei Vervollständigungen wird niemals der Piepston ausgegeben.
<i>nomatch</i>	Bei Vervollständigungsversuchen wird nur dann der Piepston ausgegeben, wenn keinerlei Vervollständigung möglich ist.
<i>ambiguous</i>	Bei Vervollständigungsversuchen wird nur dann der Piepston ausgegeben, wenn mehrere Vervollständigungen möglich sind.
<i>notunique</i>	Bei Vervollständigungsversuchen wird nur dann der Piepston ausgegeben, wenn zwar eine Vervollständigung bereits komplett erreicht wurde, aber noch weitere (längere) Vervollständigungen möglich sind.

Ist **matchbeep** nicht explizit gesetzt, entspricht dies *ambiguous*.

nobeep

Wenn gesetzt, wird niemals ein Piepston ausgegeben (siehe auch Variable **visiblebell**).

noclobber

Wenn gesetzt, ist ein Überschreiben einer existierenden Datei mit `>` bzw. `>&` nicht möglich, sondern nur mit `>!` bzw. `>&!`. Zudem gilt, wenn **noclobber** gesetzt ist, daß die Verwendung von `>>` bzw. `>>&` auf eine nicht existierende Datei nicht erfolgreich ist; in diesem Fall müßte `>>!` bzw. `>>&!` verwendet werden. Voreinstellung: nicht gesetzt.

noglob

Wenn definiert, wird die Dateinamen- und Directorystack-Substitution ausgeschaltet. Voreinstellung: nicht gesetzt.

nokanji

Wenn gesetzt und die tcsh unterstützt *Kanji*-Modus (siehe auch Variable **version**), wird dieser Modus ausgeschaltet, damit Metazeichen benutzt werden können.

nonomatch

Wenn gesetzt, wertet die tcsh ein Metazeichen für Dateinamen- und Directorystack-Substitution (`* ? [..] {..}`), das keinen Dateinamen abdeckt, nicht als Fehler. Voreinstellung: nicht gesetzt.

nostat

Enthält die Liste von Directories bzw. Wörter mit Metazeichen, die Directories abdecken, die bei Vervollständigungsversuchen ausgeschlossen werden sollen.

notify

Wenn gesetzt, wird die Beendigung von Jobs sofort und nicht erst bei der nächsten Prompt-Ausgabe gemeldet. Voreinstellung: nicht gesetzt.

oid (nur auf Domain / OS)

Enthält reale Organisations-ID des Benutzers.

owd

enthält das alte Working-Directory; ist äquivalent zum Bindestrich (-) bei **cd** und **pushd**.

path

Enthält die Suchdirectories für Programme.

Voreinstellung: Wird mit den Werten der Variablen **PATH** aus der Standardumgebung besetzt, oder falls **PATH** nicht gesetzt ist, auf eine vom System vorgegebene Voreinstellung.

Eine *tcsh*, für die weder die Option **-c** noch die Option **-t** gesetzt ist, trägt automatisch, nachdem sie *~/tcshrc* gelesen hat, alle Inhalte der in **path** angegebenen Directories in die interne Hashing-Tabelle ein. Das gleiche geschieht auch immer dann, wenn der Inhalt von **path** verändert wird. Legt man jedoch in einem der **path**-Directories ein neues Programm an, muß man **rehash** aufrufen, damit die *tcsh* dieses Kommando auch findet, also in die Hashing-Tabelle einträgt.

printexitvalue

Wenn gesetzt, wird bei Beendigung jedes interaktiven Programms dessen exit-Status (Variable **status**) ausgegeben, wenn dieser verschieden von 0 ist.

prompt

Legt den Primär-Promptstring für die *tcsh* fest.

In **prompt** sind die folgenden Format-Angaben möglich, die bei der Ausgabe entsprechend ersetzt werden:

%/ Working-Directory

%~ Working-Directory, wobei jedoch immer anstelle des eigenen Working-Directorys ~ und bei Working-Directories von anderen Benutzer *~loginname* ausgegeben wird, wobei *~loginname* nur dann ausgegeben wird, wenn der

Benutzer `~loginname` entweder in einem Pfadnamen oder sonst in der aktuellen Sitzung bereits verwendet hat.

%c[[0]n], %.[[0]n] die letzte Komponente bzw. die letzten *n* Komponenten des Working-Directorys. Beginnt die Zahl *n* mit einer **0**, wird die Zahl der führenden, nicht angezeigten Komponenten (*na*) im Format `/ <na>` vor der eigentlichen Ausgabe angezeigt. Ist die Shellvariable **ellipsis** gesetzt, werden für die führenden, nicht angezeigten Komponenten drei Punkte ... (Ellipse) ausgegeben. Bei diesem Format ist automatisch **%~** eingeschaltet, außer es wurde *n* angegeben.

```
> set prompt = "%.2 "
~ cd /usr/include/sys
include/sys set prompt = "%c "
sys set prompt = "%c02 "
/<1>include/sys set prompt = "%c01 "
/<2>sys
```

%C	Wie %c , wobei jedoch %~ ausgeschaltet ist.
%h, %!, !	Aktuelle Nummer in der History-Liste.
%M	Voller Hostname.
%m	Hostname bis zum ersten Punkt (.).
%S (%s)	Standout-Modus einschalten (ausschalten).
%B (%b)	Fettschrift einschalten (ausschalten).
%U (%u)	Unterstreichen einschalten (ausschalten).
%t, %@	Zeit im 12-Stunden-AM/PM-Format.
%T	wie %t , aber im 24-Stunden-Format (abhängig von Variable ampm).
%p	Zeit mit Sekunden im 12-Stunden-AM/PM-Format.
%P	Wie %p , aber im 24-Stunden-Format (abhängig von Variable ampm).
\c	Zeichen <i>c</i> wird wie bei <i>bindkey</i> behandelt.
^c	Zeichen <i>c</i> wird wie bei <i>bindkey</i> behandelt.
%%	Zeichen %
%n	Benutzername
%d	Wochentag (<i>Mon, Tue, Thu, ...</i>)

%D	Wochentag im Monat
%w	Monat (<i>Jan, Feb, Mar, ...</i>)
%W	Monatsnummer (<i>01, 02, 03, ...</i>)
%y	Zweistelliges Jahr (<i>96, 97, 98, ...</i>)
%Y	Vierstelliges Jahr (<i>1996, 1997, 1998, ...</i>)
%l	<i>tty</i> -Datei der Shell (<i>tty3, ...</i>)
%L	Löscht vom Promptende bis zum Zeilenende.
%#	> (oder erstes Zeichen aus Shellvariable promptchars) für normale Benutzer bzw. # (oder zweites Zeichen aus Shellvariable promptchars) für Superuser.
%{string%}	<i>string</i> als Escape-Sequenz. Damit können Terminalattribute geändert werden, wobei auf ein Bewegen des Cursors verzichtet werden sollte. Diese Angabe darf nicht am Ende von prompt stehen.
%?	Rückgabewert des Kommandos, das vor Ausgabe des Prompts ausgeführt wurde.
%R	In prompt2 der Status des Parsers. In prompt3 der korrigierte String. In history der History-String.

%B, %S, %U und **%{string%}**

sind nur in Shells verfügbar, die *eight bit clean* sind (siehe auch Variable **version**).

Die folgende Festlegung

```
set prompt = "%m [%h] %B[%T]%b %/> "
```

würde z.B. folgende Prompt-Ausgabe nach sich ziehen

```
hh [54] [10:58] /home/hh>
```

Voreinstellung für interaktive Shells ist **%#**.

prompt2

Legt den String fest, der in **while**- oder **foreach**-Schleife bzw. nach Zeilen, die mit dem Forsetzungszeichen \ enden, als Prompt auszugeben ist. Hierbei können die gleichen Formatangaben verwendet werden wie bei **prompt**. Voreinstellung für interaktive Shells ist **%R?**.

prompt3

Legt den String fest, der bei automatischen Korrekturen zur Bestätigung dieser Korrektur als Prompt auszugeben ist. Hierbei können die gleichen Formatangaben verwendet werden wie bei **prompt**. Voreinstellung für interaktive Shells ist »**CORRECT**>%R (y | n | e)? «.

promptchars

Legt für die Formatangabe %# bei der Variablen **prompt** das einzusetzende Zeichen fest: Erstes Zeichen für normale Benutzer und zweites Zeichen für Superuser.

pushdsilent

Bei jedem Aufruf von **pushd** und **popd** wird der Directorystack ausgegeben. Um diese automatische Ausgabe auszuschalten, muß diese Shellvariable gesetzt werden.

pushdthome

Wenn gesetzt, ist ein **pushd**-Aufruf ohne Argumente identisch zu **pushd ~** (wie bei **cd**).

recexact

Wenn gesetzt, wird die kürzest mögliche Wortvervollständigung akzeptiert, selbst wenn längere Vervollständigungen möglich sind.

recognize_only_executables

Wenn gesetzt, werden bei Vervollständigungsmöglichkeiten nur ausführbare Dateien aufgelistet.

rmstar

Wenn gesetzt, wird bei jedem **rm *** automatisch nachfragt, ob dies wirklich erwünscht ist.

savedirs

Wenn gesetzt, wird beim Beenden der Shell **dirs -S** ausgeführt.

savehist

Wenn gesetzt, wird beim Beenden der Shell **history -S** ausgeführt. Ist das erste Wort von **savehist** eine Zahl, werden nur so viele Kommandozeilen aus der History-Liste gesichert. Ist das zweite Wort von **savehist merge**, werden die Kommandozeilen mit den eventuell schon in der History-Datei vorhandenen Kommandozeilen gemischt, wobei sie entsprechend den Zeitmarken sortiert werden. Ein solches Mischen ist nützlich in einer Umgebung wie X Windows oder beim Arbeiten mit virtuellen Terminals, wo meist mehrere Shells gleichzeitig aktiv sind.

sched

Legt das Format fest, das vom builtin-Kommando **sched** bei der Ausgabe von momentan mit **sched** eingerichteten Kommandos verwendet wird. Voreinstellung ist »%h\t%T\t%R\n« (siehe auch Variable **prompt**).

shell

Legt den Interpreter für ausführbare Skripts fest, die nicht selbst in der ersten Zeile mit *#interpreter* ihren eigenen Interpreter auswählen. Das erste Wort in **shell** sollte ein voller Pfadname sein, wie z.B. */bin/csh* oder */usr/bin/tcsh*.

shlvl

Enthält die Schachtelungstiefe der aktuellen Shell und zeigt somit an, wie viele »Vorfahren« die aktuelle Shell hat. Bei Login-Shells wird **shlvl** auf 1 gesetzt.

showdots

Wenn gesetzt, werden beim Auflisten von möglichen Wortvervollständigungen auch Dateien angezeigt, deren Name mit Punkt (.) beginnt. Ist **showdots** auf **-A** gesetzt, werden jedoch die beiden Directories *.* und *..* nicht angezeigt.

status

Enthält den exit-Status des letzten Kommandos. Wenn dieses abnormal beendet wurde, wird auf diesen exit-Status immer *0200* (dezimal 128) addiert. Builtin-Kommandos liefern den exit-Status 0 (wenn erfolgreich) und sonst 1.

symlinks

Legt fest, was bei Zugriffe auf symbolische Links, die auf Directories zeigen, geschehen soll. Nachfolgend zunächst ein Beispiel, was passiert, wenn **symlinks** nicht gesetzt ist:

```
> unset symlinks
> cd /tmp
> mkdir von von/src nach
> ln -s ../von/src nach/dir
> cd /tmp/nach/dir; echo $cwd
/tmp/nach/dir
> cd ../; echo $cwd
/tmp/von
```

symlinks kann nun auf die folgenden Werte gesetzt werden:

chase Beim Wechseln in ein Directory, das ein symbolischer Link ist, wird zum Directory gewechselt, auf das der symbolische Link weist. Beim Home-Directory des Benutzers funktioniert dies momentan nicht.

```
> set symlinks = chase
> cd /tmp/nach/dir; echo $cwd
/tmp/von/src
> cd ..; echo $cwd
/tmp/von
```

ignore Beim Wechseln in ein Directory, das ein symbolischer Link ist, versucht die Shell ein Directory relativ zum aktuellen Directory zu konstruieren, bevor sie dem Link folgt. Bei einem folgenden **cd ..** befindet man sich dann wieder im ursprünglichen Directory. Dies funktioniert nur bei builtin-Kommandos und Dateinamen-Vervollständigung.

```
> set symlinks = ignore
> cd /tmp/nach/dir; echo $cwd
/tmp/nach/dir
> cd ..; echo $cwd
/tmp/nach
```

expand Entspricht in etwa *ignore*, nur daß es sowohl für builtin- als auch für nicht builtin-Kommandos funktioniert. Die Expandierung von Pfadnamen relativ zum aktuellen Directory kann durch Quoting ausgeschaltet werden.

```
> set symlinks = expand
> cd /tmp/nach/dir; echo $cwd
/tmp/nach/dir
> cd ..; echo $cwd [verhält sich wie ignore]
/tmp/nach
> cd /tmp/nach/dir; echo $cwd
/tmp/nach/dir
> cd ".."; echo $cwd [Expand. mit Quoting ausgeschaltet]
/tmp/von
> cd /tmp/nach/dir; echo $cwd
/tmp/nach/dir
> /bin/echo ..
/tmp/nach
> /bin/echo ".."
..
```

tcsh

Enthält die Versionsnummer der tcsh im Format *R.VV.PP* (*R*=Release-Nummer, *VV*=Versionsnummer und *PP*=Patchlevel)

term

Enthält die Terminal-Bezeichnung. Voreinstellung: Wird mit dem Wert der Variablen **TERM** aus der Standardumgebung besetzt.

time

Die in **time** angegebene Zahl legt eine Zeitmarke in CPU-Sekunden fest. Wird bei der Ausführung eines Kommandos diese Zeitmarke überschritten, wird nach Beendigung des Kommandos automatisch das builtin-Kommando **time** aufgerufen.

Wenn ein zweites Wort angegeben ist, legt dies das Format für die Ausgabe des **time**-Kommandos fest. In diesem zweiten Wort sind die folgenden Formatangaben möglich:

- %U** CPU-Sekunden, die Prozeß im *user*-Modus arbeitete.
- %S** CPU-Sekunden, die Prozeß im *kernel*-Modus arbeitete.
- %E** (*elapsed*) Uhrzeit (in Sekunden), die Prozeß aktiv war.
- %P** Prozentzahl, die das Verhältnis von (**%U** + **%S**) : **%E** anzeigt.
- %W** Zahl, die angibt wie oft Prozeß *geswappt* wurde.
- %X** Durchschnittliche KBytes für *text*-Segment.
- %D** Durchschnittliche KBytes für *data / stack*-Segment.
- %K** Gesamtzahl von benutzten KBytes (**%X** + **%D**).
- %M** Maximal belegter Speicher in KBytes.
- %F** Anzahl von *major page faults* (*page* mußte von Disk geladen werden).
- %R** Anzahl von *minor page faults*.
- %I** Anzahl von *input*-Operationen.
- %O** Anzahl von *output*-Operationen.
- %r** Anzahl von empfangenen *socket-messages*.
- %s** Anzahl von gesendeten *socket-messages*.
- %k** Anzahl von empfangenen Signalen.
- %w** Anzahl von freiwilligen *context switches (waits)*.
- %c** Anzahl von unfreiwilligen *context switches*.

tperiod

Legt das Zeitintervall (in Minuten) fest, in dem die im speziellen Alias **periodic** angegebenen Kommandos auszuführen sind (siehe auch spezielles Alias **periodic**).

tty

Enthält den *tty*-Namen.

uid

Enthält die reale user-ID des Benutzers.

user

Enthält den Loginnamen des Benutzers.

verbose

Wenn gesetzt, wird nach jeder History-Substitution das entsprechend erzeugte Kommando angezeigt, bevor es ausgeführt wird. **verbose** wird auch durch die Kommandozeilen-Option **-v** gesetzt.

version

Enthält die tcsh-Versionsnummer (siehe Variable **shell**), Ursprung, Release-Datum, Hersteller (siehe Environment-Variable **VENDOR**), Betriebssystem (siehe Environment-Variable **OSTYPE**), Maschine (siehe Environment-Variable **MACHTYPE**) und eine mit Komma getrennte Liste von Optionen, die zur Kompilierungszeit gesetzt wurden. Mögliche Optionen sind dabei:

8b Die tcsh ist *eight bit clean*; Voreinstellung.

7b Die tcsh ist nicht *eight bit clean*.

nls NLS des Systems wird verwendet.

lf Login-Shells führen */etc/csh.login* vor und nicht nach */etc/csh.cshrc* aus. Ebenso führen sie *~/login* vor und nicht nach *~/tcshrc* und *~/history* aus.

dl Punkt (.) ist aus Sicherheitsgründen als letztes in **path** angegeben; Voreinstellung.

nd Punkt (.) ist aus Sicherheitsgründen nicht in **path** angegeben.

vi **vi** (nicht **emacs**) wird als Kommandozeilen-Editor verwendet.

bye **bye** ist ein Synonym für **logout** und **log** ist ein alternativer Name für **watchlog**.

al **autologout** ist eingeschaltet; Voreinstellung.

kan *Kanji* wird verwendet und der ISO-Zeichensatz wird ignoriert, außer die Shellvariable **nokanji** ist gesetzt.

sm das *malloc* des Systems wird verwendet.

hb Beim Ausführen von Shellskripts wird die Konvention *#!programm argument(e)* emuliert.

ng Builtin-Kommando **newgrp** ist verfügbar.

rh Die Shell versucht die Environment-Variable **REMOTEHOST** zu setzen.

visiblebell

Wenn gesetzt, dann wird anstelle des Piepstones ein visuelles Signal (wie Bildschirm blinken lassen) ausgegeben.

watch

Enthält eine Liste von Benutzer/Terminal-Paaren, die ständig auf An- bzw. Abmeldungen hin beobachtet werden sollen. Ist in dieser Liste anstelle eines Benutzers oder eines Terminal *any* angegeben, werden An- bzw. Abmeldungen aller Benutzer oder Terminals überwacht und gemeldet. z.B. bewirkt die folgende Angabe

```
set watch = (hans tty3 any console $user any)
```

daß eine An- bzw. Abmeldung des Benutzers *hans* am Terminal *tty3*, alle An- und Abmeldungen an der Konsole und alle eigenen An- bzw. Abmeldungen an beliebigen Terminals gemeldet werden. Die folgende Angabe

```
set watch = ('h*' any any tty4)
```

bewirkt, daß jede An- bzw. Abmeldung (an einem beliebigen Terminal) eines Benutzers, dessen Loginname mit *h* beginnt und alle An- bzw. Abmeldungen am Terminal *tty4* gemeldet werden.

An- und Abmeldungen werden dabei in einem Zeitintervall von 10 Minuten überprüft. Ist das erste Wort von **watch** jedoch eine Zahl, legt diese ein neues Minuten-Intervall für die Überprüfung fest. So würde z.B.

```
set watch = (1 any any)
```

jede Minute über alle neuen An- und Abmeldungen informieren.

Möchte man zu einem beliebigem Zeitpunkt die aktuellen Anmeldungen erfahren, bevor das entsprechende Zeitintervall abgelaufen ist, muß das builtin-Kommando **log** aufrufen.

Die Shellvariable **who** legt das Format von **watch**-Ausgaben fest.

who

Enthält die Formatangaben für **watch**-Meldungen. Es sind folgende Formatangaben möglich:

- %n** Name des Benutzers, der sich an- bzw. abgemeldet hat.
- %a** beobachtete Aktion, wie z.B. *logged on*, *logged off* oder *replaced ... on*.
- %l** Terminal (tty), an dem sich Benutzer an- bzw. abgemeldet hat.
- %M** Voller Hostname oder *local*, wenn es sich um einen lokalen Host handelt.
- %m** Voller Hostname bis zum ersten Punkt (.) oder *local*, wenn es sich um einen lokalen Host handelt.

%M und **%m** sind nur auf Systemen verfügbar, die den Hostnamen in */etc/utmp* angegeben haben. Wird **who** nicht explizit gesetzt, hat es die Voreinstellung »%n has %a %l from %m.« oder »%n has %a %l.« (auf Systemen, die Hostnamen nicht in */etc/utmp* angegeben haben).

wordchars

Enthält eine Liste von nicht-alphanumerischen Zeichen, die als Bestandteil eines Wortes bei den Editor-Kommandos *forward-char*, *backward-char* usw. interpretiert werden. Wenn nicht gesetzt, entspricht dies der Angabe:

set wordchars = '*?_-.[]~='.

8.14 Environment

Die tcsh bietet folgende Environment-Variablen an:

AFSUSER	Äquivalent zur Shellvariable afsuser .
COLUMNS	Anzahl der Spalten auf dem Terminal.
DISPLAY	Wird von X Windows benutzt; ist DISPLAY gesetzt, setzt die tcsh autologout nicht.
EDITOR	Legt den einzuschaltenden builtin-Editor fest, wenn Variable VISUAL nicht gesetzt ist (siehe auch Environment-Variable VISUAL und Editor-Kommando <i>run-fg-editor</i>).
GROUP	Äquivalent zur Shellvariable group .
HOME	Äquivalent zur Shellvariable home .
HOST	Enthält den Namen der Maschine, auf der die tcsh abläuft.
HOSTTYPE	Enthält den Typ der Maschine, auf der die tcsh abläuft. Diese Variable ist überflüssig, und wird in zukünftigen Freigaben nicht mehr angeboten werden.
HPATH	Enthält eine Liste von Directories (mit Komma getrennt), die beim Editor-Kommando <i>run-help</i> nach Dokumentations-Dateien zu durchsuchen sind.
LANG	Spezifiziert länderspezifischen Zeichensatz.
LC_CTYPE	Spezifiziert das Verhalten von Funktionen aus <i><ctype.h></i> .
LINES	Anzahl der Zeilen auf dem Terminal.
MACHTYPE	Enthält Maschinentyp (Mikroprozessor, ...).
NOREBIND	Wenn gesetzt, werden nicht verwendete Zeichen nicht automatisch dem Editor-Kommando <i>self-insert-command</i> zugeteilt.

OSTYPE	Enthält den Namen des Betriebssystems.
PATH	Äquivalent zur Shellvariable path . Die einzelnen Directories sind hier nicht mit Leerzeichen, sondern mit Doppelpunkten (:) getrennt.
PWD	Äquivalent zur Shellvariable cwd , jedoch nicht mit dieser synchronisiert.
REMOTEHOST	Der Host, von dem sich der Benutzer aus angemeldet hat.
SHLVL	Äquivalent zur Shellvariable shlvl .
SYSTYPE	(nur auf Domain / OS) enthält den Systemtyp.
TERM	Äquivalent zur Shellvariable term .
TERMCAP	Enthält den <i>terminal capability string</i> .
USER	Äquivalent zur Shellvariable user .
VENDOR	Enthält den Namen des Herstellers.
VISUAL	Legt den einzuschaltenden builtin-Editor fest (siehe auch Environment-Variable EDITOR und Editor-Kommando <i>run-fg-editor</i>).

8.15 Anwendungsbeispiel

Hier wird ein tcsh-Skript vorgestellt, das das Acht-Damen-Problem löst. Beim Acht-Damen-Problem gilt es, acht Damen so auf einem Schachbrett zu positionieren, daß keine der Damen eine andere bedroht. Das nachfolgende tcsh-Skript gibt alle Lösungen aus.

```
#!/bin/tcsh
# achtdame - loest das Achtdame-Problem mit Backtracking
#           Skript sehr zeitaufwendig.

if ($#argv == 0) then
    set spalte = ( 0 0 0 0 0 0 0 0 )
    set spaltfrei = ( 1 1 1 1 1 1 1 1 )
    set aufwaertsfrei = ( 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 )
    set abwaertsfrei = ( 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 )
    set zeile = 0
    echo "0" >.lsg
else
    set spalte = ($argv[1])
    set spaltfrei = ($argv[2])
    set aufwaertsfrei = ($argv[3])
    set abwaertsfrei = ($argv[4])
    set zeile = $argv[5]
endif
```

```
@ zeile++
@ s = 1
while ($s <= 8)
    @ z = $zeile + $s
    @ zx = $zeile + 8 - $s
    if ($spaltfrei[$s] && $aufwaertsfrei[$z] && $abwaertsfrei[$zx]) then
        set spalte[$zeile] = $s
        set spaltfrei[$s] = 0
        set aufwaertsfrei[$z] = 0
        set abwaertsfrei[$zx] = 0
        if ( $zeile == 8 ) then
            set lsg = `cat .lsg`
            @ lsg++
            echo "$lsg" >.lsg
            echo "${lsg}.Loesung:"
            echo " "
            @ i = 1
            while ($i <= 8)
                echo "|---|---|---|---|---|---|---|---|"
                @ j = 1
                while ($j <= 8)
                    if ($spalte[$j] == $i) then
                        echo "| X \c"
                    else
                        echo "| \c"
                    endif
                    @ j++
                end
                echo "|"
                @ i++
            end
            echo "|---|---|---|---|---|---|---|---|"
            echo "\n\nWeiter mit Return-Taste ....."
            set xx = $<
        else
            # Rekursiver Aufruf des Skripts
            # Option -f legt fest: .cshrc nicht lesen
            csh -f $0 "$spalte" \
                "$spaltfrei" \
                "$aufwaertsfrei" \
                "$abwaertsfrei" \
                "$zeile"
        endif
        set spaltfrei[$s] = 1
        set aufwaertsfrei[$z] = 1
        set abwaertsfrei[$zx] = 1
    endif
    @ s++
end
@ zeile--
exit 0
```


9 Die beiden Tischrechner **dc** und **bc**

*Non statim pusillum est si quid maximo minus est.
(Wenn etwas kleiner ist als das Größte, ist es darum noch lange nicht unbedeutend.)*

Seneca

Die beiden Kommandos **dc** und **bc** simulieren einen Rechner, der arithmetische Berechnungen mit beliebiger Genauigkeit durchführen kann. So sind sie in der Lage, z.B. zwei Zahlen mit jeweils 500 Ziffern zu multiplizieren und das Ergebnis (eine tausendstellige Zahl) innerhalb von wenigen Sekunden auszugeben.

Während **dc** die zu berechnenden arithmetischen Ausdrücke in der doch ungewohnten Postfix-Schreibweise (auch umgekehrte polnische Notation genannt) entgegennimmt, arbeitet **bc** mit der allgemein üblichen Infix-Schreibweise.

```
$ bc
14 + 9
23
[Strg-D]
$ dc
14 9 +
p
23
[Strg-D]
$
```

[Ergebnis wird an oberster Stelle eines Stacks gespeichert]
[print: Gib oberstes Stack-Element aus]

Bei **bc** handelt es sich also um einen wesentlich bedienerfreundlicheren Rechner. Allerdings verwendet **bc** den Rechner **dc**, um die eigentlichen Berechnungen durchführen zu lassen. Dazu wandelt **bc** die ihm übergebenen Ausdrücke in die **dc**-Notation um, um sie dann von **dc** berechnen zu lassen. **bc** ist also ein Präprozessor oder anders ausgedrückt, eine verbesserte Benutzeroberfläche zum Rechner **dc**.

9.1 Der Tischrechner **dc**

dc (*desk calculator*) ist ein Tischrechner, der arithmetische Berechnungen mit beliebiger Genauigkeit durchführt. Normalerweise arbeitet er im Zehnersystem, allerdings kann sowohl für die Eingabe als auch für die Ausgabe von Zahlen ein anderes Zahlensystem gewählt werden.

dc arbeitet als Stack-Maschine. Dies bedeutet, daß die zu berechnenden Ausdrücke in Postfix-Schreibweise (umgekehrte polnische Notation) einzugeben sind.

durchgeführt. Nach dieser Berechnung werden diese beiden obersten Stack-Werte entfernt und dann das Ergebnis der Berechnung an oberster Stelle des Stacks gespeichert. Als Exponent (bei $\wedge$) dürfen nur ganze Zahlen angegeben werden.

Bei Eingabe von **v** gilt folgendes:

- v** der oberste Stack-Wert wird durch seine Quadratwurzel ersetzt. Für die Quadratwurzel werden dabei so viele Nachkommastellen berechnet, wie in der ursprünglichen Zahl vorhanden sind. Das heißt z.B., daß für die Quadratwurzel einer ganzen Zahl keine Nachkommastellen berechnet werden.

Beispiel

```
$ dc
4.2 5 +
6*p
55.2
[Strg-D]
$
```

Erklärung:

Bei (1) ergeben sich folgende Aktionen im Stack:

```
+-----+ (oberstes
| 5 | Stack-Element)
+-----+ -----> +-----+ (oberstes
| 4.2 | + | 9.2 | Stack-Element)
+-----+ +-----+
```

Mit (2) ergeben sich dann folgende Aktionen im Stack:

```
+-----+ (oberstes
| 6 | Stack-Element)
+-----+ -----> +-----+ (oberstes
| 9.2 | * | 55.2 | Stack-Element; wird mit p
+-----+ +-----+ ausgegeben)
```

```
$ dc
4 5 6+*p
44
4 5+6*p
54
[Strg-D]
$ dc
3.3 4^p
118.5
3 3^3^p
19683 [Ergebnis von: 27 hoch 3]
3 3 3^^p
7625597484987 [Ergebnis von: 3 hoch 27]
```

```
[Strg-D]
$ dc
2.34vp
1.52
200vp
14                                [keine Berücksichtigung der Nachkommastellen]
200.0vp
14.1                             [Berechnung einer Stelle nach dem Komma]
200.000000vp
14.142135                       [Berechnung auf 6 Stellen nach dem Komma]
$
```

Push-, Pop- und Duplizier-Operationen

sr (*store*) Das oberste Stack-Element wird entfernt (pop-Operation) und im Register *r* gespeichert. Für *r* kann dabei ein beliebiges Zeichen angegeben werden.

Wird für *s* das große **S** angegeben, wird *r* als ein Stack (nicht Register) interpretiert und das oberste Element des Haupt-Stacks an oberster dieses Stacks *r* gespeichert.

lr (*load*) Der Wert des Registers *r* wird an oberster Stelle des Stacks gespeichert (push-Operation). Der Inhalt des Registers *r* wird durch die Operation nicht verändert. Alle Register sind mit 0 initialisiert.

Wird für *l* das große **L** angegeben, wird für *r* ein Stack angenommen, dessen oberstes Element auf den Haupt-Stack verlagert wird.

d (*duplicate*) Der oberste Wert des Stacks wird dupliziert.

Beispiel

```
$ dc
4 5*6 7*+p      [ (4*5) + (6*7) ]
62
[Strg-D]
$ dc
4 5*
sa              [Speichere 20 im Register a]
6 7*
la              [Stack beinhaltet nun einen Wert: 42 ]
*p
840             [Stack enthält nun von oben nach unten: 20 42 ]
[Strg-D]
$ dc
4 20
Sa
Sa              [Der Stack a enthält nun von oben nach unten: 4 20;
                Hauptstack ist leer ]
6 7*
La              [Hauptstack beinhaltet nun einen Wert: 42 ]
La              [Hauptstack enthält nun von oben nach unten: 20 4 42 ]
```

```
*
+
p
122
[Strg-D]
$
```

[Hauptstack enthält nun von oben nach unten: 80 42]
[Hauptstack enthält nun nur noch einen Wert: 122]

Die zuletzt gegebenen Anweisungen hätte man auch alle in einer Zeile angeben können:

```
$ dc
4 20SaSa6 7*LaLa*+p
122
[Strg-D]
$
```

```
$ dc
3 5*d
+p
30
[Strg-D]
$
```

[Stack enthält zweimal den gleichen Wert: 15 15]

Eingabebasis, Ausgabebasis und Skalierungsfaktor

dc arbeitet normalerweise im Zehnersystem (Basis=10), allerdings kann er auch in anderen Zahlensystemen arbeiten. Da **dc** zwischen der Zahlen-Eingabe und -Ausgabe unterscheidet, ist es dem Benutzer möglich, sowohl für die Eingabe als auch für die Ausgabe ein eigenes Zahlensystem festzulegen. Deshalb kann **dc** auch sehr einfach für Zahlenkonvertierungen eingesetzt werden.

Daneben bietet **dc** auch noch die Möglichkeit, einen Skalierungsfaktor festzulegen. Der Skalierungsfaktor gibt an, wie viele Nachkommastellen **dc** bei seinen Berechnungen berücksichtigen soll.

- i** (*input base*) Der oberste Wert des Stacks wird entfernt und als neue Eingabe-Basis für **dc** verwendet.
- I** (*Input base*) Die momentane Eingabe-Basis wird an oberster Stelle des Stacks gespeichert.
- o** (*output base*) Der oberste Wert des Stacks wird entfernt und als neue Ausgabe-Basis für **dc** verwendet.
- O** (*Output base*) Die momentane Ausgabe-Basis wird an oberster Stelle des Stacks gespeichert.
- k** Der oberste Wert des Stacks wird entfernt und als nicht-negativer Skalierungsfaktor verwendet. Die Voreinstellung für den Skalierungsfaktor ist 0.

- K** Der momentane Skalierungsfaktor wird an oberster Stelle des Stacks gespeichert.
- X** (**eXchange**) Die an oberste Stelle des Stacks stehende Zahl wird durch ihren Skalierungsfaktor ersetzt.

Beispiel

```
$ dc
132
2o [Ausgabe-Basis ab jetzt 2 (Dualsystem)]
p [Gib obersten Stack-Wert (132) dual aus]
10000100
14.1
p [Gib obersten Stack-Wert (14.1) dual aus]
1110.000
16o [Ausgabe-Basis ab jetzt 16 (Hexadezimalsystem)]
p [Gib obersten Stack-Wert (14.1) hexadezimal aus]
E.1
[Strg-D]
$

$ dc
1 3/ [Dividiere 1 durch 3]
p
0 [Skalierungsfaktor auf 0 (Nachkommastellen)
voreingestellt]
10k [Skalierungsfaktor ab jetzt 10 (Nachkommastellen)]
p
0 [Berechnung von 1/3 bereits durchgeführt;
Nach-Skalierung nicht möglich]
1 3/ [Berechne 1/3 neu]
p
0.3333333333
[Strg-D]
$

$ dc
2i [Eingabe-Basis ab jetzt 2 (Dualsystem)]
110110.000000010101
p
54.004638671875
1010 1010^ [Entspricht dezimal: 10 hoch 10]
p
10000000000
[Strg-D]
$
```

dc-Programme

Es ist möglich, ganze **dc**-Programme in Form eines Strings anzugeben. Solche Strings können an oberster Stelle des Stacks oder in einem Register gespeichert werden und dann zur Ausführung gebracht werden.

- [string]** Der in eckigen Klammern angegebene *string* wird an oberster Stelle des Stacks gespeichert.
- x** (*execute*) Das oberste Stack-Element wird als ein String von **dc**-Kommandos interpretiert, die ausgeführt werden.
- <r >r =r** Die zwei obersten Stack-Elemente werden verglichen. Trifft die angegebene Bedingung (<, > bzw. =) zu, werden die in Register *r* als String gespeicherten **dc**-Kommandos ausgeführt. Soll die angegebene Bedingung negiert werden, ist dem jeweiligen Vergleichsoperator ein ! voranzustellen: **!<r !>r !=r**.

Beispiel

```
$ dc
[1bp2+sb1b20>a]sa
1sb 1ax          [Diese beiden Zeilen bewirken Ausgabe der ungeraden
                  Zahlen von 1 bis 19]
1
3
5
7
9
11
13
15
17
19
[1a1+dsa*pla10>y]sy
0sa1
1yx             [Diese drei Zeilen bewirken Ausgabe aller Fakultäten
                  zw. 1 und 10]
1
2
6
24
120
720
5040
40320
362880
3628800
[Strg-D]
$
```

Ausgaben

- p** (**print**) Der oberste Wert des Stacks wird ausgegeben, aber nicht verändert.
- P** (**Print**) Der oberste Wert des Stacks wird als String (**dc**-Programm) interpretiert, vom Stack entfernt und ausgegeben.
- f** Alle Werte des Stacks werden ausgegeben.

Beispiel \$ dc ↵
 7 6 5 4 3 2 1 ↵
 f ↵
 1
 2
 3
 4
 5
 6
 7
 * ↵
 f ↵
 2
 3
 4
 5
 6
 7
 *** ↵
 f ↵
 120
 6
 7
 sa ↵
 f ↵
 6
 7
 * ↵
 f ↵
 42
 la ↵
 f ↵
 120
 42
 * ↵
 f ↵
 5040
 \$

Verlassen von **dc**

- q** (**quit**) Verlassen von **dc**.
 Wenn gerade ein String von **dc**-Kommandos ausgeführt wird, werden zwei Rekursions-Ebenen bei dieser String-Ausführung verlassen.
- Q** (**Quit**) Verlassen von **dc**.
 Wenn gerade ein String von **dc**-Kommandos ausgeführt wird, wird der momentan oberste Stack-Wert entfernt. Dieser Wert legt dabei die Anzahl von Rekursions-Ebenen fest, die bei dieser String-Ausführung zu verlassen sind.

Sonstiges

- z** Die Stack-Größe wird an oberster Stelle des Stacks gespeichert.
- Z** Das oberste Stack-Element wird durch seine Länge ersetzt.
- c** (**clear**) Der gesamte Stack wird geleert.
- !** Der Rest der Zeile wird als UNIX-Kommando interpretiert und von der Shell zur Ausführung gebracht.
- ?** Eine Eingabezeile wird von der Dialogstation gelesen.
- ::** werden von **bc** für Array-Operationen verwendet.
 - :x** entfernt das oberste Stack-Element und verwendet es als Index für das Array *x*. Das nächste Element des Stacks wird dann bei diesem Index im Array *x* abgespeichert. Ein Index muß dabei größer oder gleich 0, aber kleiner als 2048 sein.
 - ;x** lädt einen Wert aus dem Array *x* in den Hauptstack. Der oberste Stack-Wert legt dabei den Index für den aus dem Array *x* zu ladenden Wert fest.

Beispiel

```
$ cat qsum.dc ␣
!echo "Gib Wert ein"
?
sa
la
Z
!echo "Ziffern-Zahl:"
p
c
[la10%+la10/dsa0<y]sy
Olyx
!echo "Quersumme:"
p
!echo "Verlasse Programm mit EOF (Strg-D) oder Eingabe von q"
$ dc qsum.dc ␣
Gib Wert ein
!
12345678901234567890 ␣
Ziffern-Zahl:
!
20
Quersumme:
!
90
Verlasse Programm mit EOF (Strg-D) oder Eingabe von q
!
q ␣
$
```

9.1.2 *dc*-Fehlermeldungen

x is unimplemented

wenn *x* eine Oktal-Zahl ist.

stack empty

wenn für eine geforderte Operation nicht genügend Stack-Elemente vorhanden sind.

Out of space

wenn nicht genügend Speicherplatz vorhanden ist (z.B. eine Zahl mit zu vielen Ziffern).

Out of headers

wenn mit zu vielen Zahlen gleichzeitig gearbeitet wird.

Out of pushdown

wenn der Stack voll ist.

Nesting Depth

wenn die maximal zugelassene Schachtelungstiefe bei Kommando-Ausführungen überschritten wird.

9.2 Der Tischrechner *bc*

bc ist – wie *dc* – ein Tischrechner, der arithmetische Berechnungen mit beliebiger Genauigkeit durchführt.

Bei *bc* handelt es sich um einen Compiler, der *bc*-Eingaben in eine dem Rechner *dc* verständliche Sprache übersetzt, um sie dann von ihm berechnen zu lassen. Während die Syntax für die von *dc* definierte Sprache äußerst umständlich und ungewohnt ist, lehnt sich die Syntax der *bc*-Sprache an die Programmiersprache C an. So kennt *bc* beispielsweise Ablaufstrukturen (wie *if*-Anweisung oder *while*-Schleife) und läßt die Definition von Funktionen zu, die dann an späterer Stelle aufgerufen werden können. Auch nimmt *bc* die Eingabe von arithmetischen Ausdrücken in der üblichen Infix-Schreibweise und nicht wie *dc* in der ungewohnten Postfix-Notation entgegen. *bc*-Programme sind deshalb auch um ein vieles lesbarer und verständlicher als vergleichbare *dc*-Programme.

9.2.1 Aufrufsyntax für *bc*

Die Aufrufsyntax für den Rechner *bc* ist:

bc [-c] [-l] [datei(en)]

Wenn *datei(en)* angegeben sind, liest *bc* zunächst den Inhalt dieser Dateien, bevor er weitere Eingaben von der Standardeingabe liest.

Sind keine *datei(en)* angegeben, liest *bc* von Beginn an von der Standardeingabe.

bc beendet sich, wenn er **EO** (**Strg-D**) oder **quit** an der Standardeingabe liest.

bc ist eigentlich ein Präprozessor für den Rechner **dc**, welchen er automatisch aufruft, nachdem er die **bc**-Anweisungen in die **dc**-Notation umgeformt hat.

Als Optionen sind erlaubt:

- **c** (*compile only*) Die für **dc** bestimmte Ausgabe wird nicht zur Ausführung an **dc** weitergeleitet, sondern auf die Standardausgabe geschrieben.
- **l** (*library*) Aus einer mathematischen Bibliothek werden bestimmte Funktionen (wie z.B. Sinus- oder Logarithmus-Funktion) geladen, die dann nachfolgend aufgerufen werden können.

9.2.2 Einfache Rechnungen mit ganzen Zahlen

Die einfachste Anweisung an **bc** ist die Eingabe eines ganzzahligen arithmetischen Ausdrucks:

```
$ bc
34+19
53
1234*81728
100852352
[Strg-D]
$
```

Es können folgende Operatoren verwendet werden:

- + Addition
- Subtraktion
- * Multiplikation
- / Division
- % Modulo-Berechnung
- ^ Potenzierung

Eine Division von zwei ganzen Zahlen liefert wieder eine ganze Zahl; eventuell entstandene Nachkommastellen werden dabei abgeschnitten. Eine Division durch 0 resultiert immer in einer Fehlermeldung:

```
$ bc
2^3
8
9/5
1
7/0
divide by 0
0
quit
$
```

Negative Zahlen werden wie üblich durch Voranstellen eines Minuszeichens gekennzeichnet:

```
$ bc
14+-5
9
[Strg-D]
$
```

Die Prioritäten der Operatoren (in absteigender Reihenfolge¹) sind:

```
( )      (1)
^        (2)
*  %    (3)
+  -    (4)
```

Klammerung muß immer dann verwendet werden, wenn eine andere Auswertungsreihenfolge erwünscht ist, als durch die Prioritäten vorgegeben ist. Während bei gleicher Priorität die Operatoren (3) und (4) »von links nach rechts« ausgewertet werden, werden Potenz-Operatoren »von rechts nach links« ausgewertet:

```
$ bc
2^3^4          [entspricht: 2^(3^4) = 2^81; Auswertung erfolgt von
                rechts nach links]
2417851639229258349412352
2^(3^4)
2417851639229258349412352
(2^3)^4        [entspricht: 8^4]
4096
8+4*5
28
(8+4)*5
60
8*5/4          [entspricht: (8*5)/4; Auswertung erfolgt von links nach
                rechts]
10
[Strg-D]
$
```

b bietet 26 interne Speicher-Register an. Die Namen dieser Register sind die Kleinbuchstaben $a, b, c, \dots, z^2$. Der Wert eines Ausdrucks kann in einem solchen Register (Variable) gespeichert werden, indem der Ausdruck mit dem Zuweisungsoperator = einer Register-Variablen zugewiesen wird.

1. Operatoren in einer Zeile besitzen gleiche Prioritäten.
2. Keine Umlaute oder ß.

Wird ein Ausdruck (bestehend aus Zahlen-Konstanten und Register-Variablen) ohne den Zuweisungsoperator angegeben, wird das Ergebnis der Ausdrucks-Auswertung ausgegeben:

```
$ bc
b=4
b
4
b=b+17
b
21
[Strg-D]
$
```

Der Inhalt von Variablen kann auch unter Verwendung der von C her bekannten Postfix- und Präfixoperatoren `++` und `--` um 1 inkrementiert bzw. dekrementiert werden.

Präfix-Schreibweise

- `+ x` Auf den Wert von `x` wird 1 addiert; der Wert des Ausdrucks ist der inkrementierte Wert.
- `- x` Vom Wert von `x` wird 1 subtrahiert; der Wert des Ausdrucks ist der dekrementierte Wert.

Postfix-Schreibweise

- `x++` Auf den Wert von `x` wird 1 addiert; der Wert des Ausdrucks ist der alte Wert von `x`.
- `x--` Vom Wert von `x` wird 1 subtrahiert; der Wert des Ausdrucks ist der alte Wert von `x`.

```
$ bc
a=5
a++
```

[Ergebnis des Ausdrucks ist 5; erst nach der Ausgabe wird inkrementiert]

```
5
a
6
b=a++
b
6
a
7
c=5
b= ++c
```

[zwischen `=` und `++` muß mindestens ein Leerzeichen angegeben sein]

```
b
6
```



```
c ↵
6
[Strg-D]
$
```

Um die Anzahl der Ziffern einer ganzen Zahl zu bestimmen, steht die builtin-Funktion **length** zur Verfügung:

```
$ bc ↵
length(1234567890) ↵
10
a=25632 ↵
b=4789994 ↵
l=length(a*b) ↵
l ↵
12
a*b ↵
122777126208
[Strg-D]
$
```

Die Quadratwurzel einer Zahl kann mit der builtin-Funktion **sqrt** berechnet werden. Für eine ganze Zahl liefert **sqrt** das Ergebnis wieder eine ganze Zahl (Nachkommateil wird abgeschnitten), wenn dies nicht mit der builtin-Variablen **scale** (siehe unten) anders festgelegt wird:

```
$ bc ↵
sqrt(15) ↵
3
a=sqrt(1243) ↵
a ↵
35
sqrt(sqrt(15)+a) ↵
6
[Strg-D]
$
```

9.2.3 Rechnungen mit reellen Zahlen

Das Syntaxdiagramm aus Abbildung 9.1 legt fest, wie reelle Zahlen angegeben werden dürfen:

Als Operatoren für reelle Zahlen sind dabei wieder die zuvor bei den ganzen Zahlen vorgestellten Operatoren zugelassen. Für den Potenzoperator gilt lediglich die Einschränkung, daß die Potenz ganzzahlig sein muß. Auch gelten die zuvor angegebenen Prioritäten.

Die Postfix- und Präfix-Notation für die beiden Operatoren **+** und **-** kann auch auf reelle Variablen angewendet werden.

```
$ bc
5.83^4
1155.24
2.3%1.1
0.1
p=3.1415
r=4.2
f=r^2*p
u=2*r*p
f
55.2904
u
26.3886
u++
26.3886
u
27.3886
--f
54.2904
f
54.2904
[Strg-D]
$
```

Folgende Funktionen stehen für reelle Zahlen zur Verfügung¹:

- s q A) (** liefert die Quadratwurzel zu A. Die Nachkommastellen des Ergebnisses werden dabei entweder durch A oder durch den Wert der builtin-Variablen **s c a** (siehe unten) festgelegt, je nachdem was mehr Nachkommastellen liefert.
- l e n g A) h (** liefert die Gesamtzahl der signifikanten Ziffer von A.
- s c a A) (** liefert die Anzahl der Nachkommastellen von A.

```
$ bc
sqrt(4.567)
2.137
sqrt(4.56700000)
2.13705404
a=sqrt(12.345139)
a
3.513564
length(a)
7
scale(a)
6
[Strg-D]
$
```

1. A steht dabei für einen Ausdruck, dessen Berechnung eine reelle Zahl liefert.

9.2.4 Skalierungsfaktor

Der Skalierungsfaktor legt fest, wie viele Nachkommastellen bei der Berechnung von reellen Ausdrücken zu berücksichtigen sind.

Über die builtin-Variable **scale** kann der Skalierungsfaktor festgelegt werden. Der Wert von **scale** muß im Bereich 0..99 liegen. Es sind also maximal bis zu 99 Nachkommastellen erlaubt. **scale** kann dabei wie jede andere Variable verwendet werden. Die Voreinstellung für **scale** ist 0.

```
$ bc
scale
0
1/3
0
scale=scale+10
1/3
.3333333333
++scale
11
1/3
.33333333333
scale=2
2/3
.66
[Strg-D]
$
```

Wenn reelle Ausdrücke mit einem Operator verknüpft werden, gelten folgende Regeln für den Skalierungsfaktor des Ergebnisses:

$a+b$, $a-b$ Der größte Skalierungsfaktor der beiden Operanden a und b legt den Skalierungsfaktor für das Ergebnis fest:

```
$ bc
2.7+3
5.7
2.2+6.1234567890
8.3234567890
a=23.5-12.2345456
b=5.1
a+b
16.3654544
quit
$
```

$a*b$ Für den Skalierungsfaktor des Ergebnisses gelten zunächst folgende Einschränkungen:

- niemals kleiner als der größte Skalierungsfaktor der beiden Operanden a und b , und
- niemals größer als die Summe der Skalierungsfaktoren der beiden Operanden a und b .

Unter Berücksichtigung dieser Einschränkungen wird – wenn möglich – der Skalierungsfaktor des Ergebnisses durch die builtin-Variable **scale** festgelegt, so daß gilt¹: $\min(s_a+s_b, \max(\text{scale}, s_a, s_b))$

```
$ bc
3.4*4.23
14.38 [Skalierungsfaktor von sb (4.23)]
scale=4
3.4*4.23
14.382 [Skalierungsfaktor: sa+sb]
3.456*4.23
14.6188 [Skalierungsfaktor: scale]
scale=90
3.456*4.23
14.61888 [Skalierungsfaktor: sa+sb]
quit
$
```

a/b Skalierungsfaktor für das Ergebnis wird durch die builtin-Variable **scale** festgelegt:

```
$ bc
3.4/4.23
0
scale=1
3.4/4.23
0.8
scale=10
3.4/4.23
0.8037825059
3/4
0.7500000000
quit
$
```

1. s_x ist dabei Skalierungsfaktor für Operand x .

$a\%b$

entspricht der Rechnung: $a - a/b * b$.

Skalierungsfaktor für das Ergebnis ist: $\max(\text{scale} + s_b, \max(s_a, s_b))$

```
$ bc
2.3%1.1
0.1
2.300%1.1
0.100 [Skalierungsfaktor: sa]
2.3%1.100
0.100 [Skalierungsfaktor: sb]
scale=5
2.3%1.1
.000010 [Skalierungsfaktor: scale+sb --> (5+1=6)]
a=2.3
b=1.1
a%b
.000010 [Skalierungsfaktor: scale+sb --> (5+1=6)]
x=a/b [a%b wird nun nach obiger Formel
nachgerechnet]

x
2.09090
x*b
2.29999
a-x*b
.00001
2.333%1.1
.000010 [Skalierungsfaktor: scale+sb --> (5+1=6)]
2.3%1.100
.00001000 [Skalierungsfaktor: scale+sb --> (5+3=8)]
2.333333333%1.1
.000002333 [Skalierungsfaktor: sb --> 9]
[Strg-D]
$
```

a^b

Skalierungsfaktor für das Ergebnis wird dabei so bestimmt, als würden die geforderten Multiplikationen bzw. Divisionen (bei negativer Potenz) nacheinander ausgeführt¹: $\min(s_a * b, \max(\text{scale}, s_a))$

```
$ bc
2.1^5
40.8
2.15^5
45.94
2.15^-5
0 [findet Division statt; es wird Wert von
scale (=0) genommen]

2.15^0
```

1. b ist der Absolutwert der angegebenen Potenz.

```

1
scale=20
2.15^5
45.9401384375
2.15^-5
.02176745726094112839
[Strg-D]
$

```

`sqrt(a)` Skalierungsfaktor für das Ergebnis ist: $\max(s_a, \text{scale})$

```

$ bc
sqrt(2)
1
sqrt(2.0)
1.4
sqrt(2.1234)
1.4571
scale=10
sqrt(2)
1.4142135623
sqrt(2.1234)
1.4571890748
quit
$

```

Bei allen internen Berechnungen werden die nicht geforderten Nachkommastellen abgeschnitten. Das heißt, daß niemals eine Rundung stattfindet.

9.2.5 Zahlenkonvertierungen und Rechnen in Zahlensystemen

Der Rechner **bc** arbeitet normalerweise im Zehnersystem (Basis=10). Allerdings kann er auch in anderen Zahlensystemen arbeiten. Um ihn auf andere Zahlensysteme umzuschalten, stehen die beiden builtin-Variablen **ibase** und **obase** zur Verfügung:

ibase (input base) legt die Basis für die Eingabe fest.
obase (output base) legt die Basis für die Ausgabe fest.

```

$ bc
ibase=2
1101101
109
1100+1010

```

[1101101 (Dual) entspricht 109 im Zehnersystem]

22 [Ergebnis von 1100+1010 (Dual) entspricht 22 im Zehnersystem]

1100*1010

120 [Ergebnis von 1100*1010 (Dual) entspricht 120 im Zehnersystem]

quit

\$

Vorsicht ist geboten, wenn die Eingabebasis verändert wurde und später dann eine neue Ein- oder Ausgabebasis eingestellt werden soll. In diesem Fall muß die neue Basis als entsprechende Zahlendarstellung in der gerade gültigen Eingabebasis eingegeben werden:

\$ bc

ibase=8

1772

1018 [1772 (Okta1) entspricht 1018 im Zehnersystem]

ibase=10

1772

1018 [Eingabebasis immer noch 8 (nicht 10)]

ibase=12 [Nun wird die Eingabebasis auf 10 gesetzt]

1772

1772

ibase=16

AFFE*2

90108

ibase=A [Eingabebasis auf 10 zurückstellen]

ibase

10

12*11

132

obase=2

ibase=2

1100*1010

1111000

111^100

100101100001

quit

\$

Es ist zwar erlaubt, **ibase** negative Zahlen oder sehr große ganze Zahlen zu zuweisen, dies bewirkt aber keine Veränderung der Eingabebasis. **ibase** sollten sinnvollerweise nur Werte aus dem Bereich 2..16 zugewiesen werden.

1. Die oktale Zahl 10 entspricht dezimal der Zahl 8, so daß diese Anweisung keine Veränderung der Eingabebasis bewirkt, sondern diese weiterhin auf 8 eingestellt bleibt.

Im Gegensatz zur Eingabebasis gibt es für die Ausgabebasis keine Begrenzung. So sind z.B. sehr große Ausgabebasen erlaubt, was für manche Anwendungsfälle sogar sinnvoll sein kann. Wenn beispielsweise **o b a d e e** Wert 1000 zugewiesen wird, werden die Zahlen in Dreier-Gruppen ausgegeben.

```
$ bc
ibase=16
2000
7D0
ibase=1000
a=6243.40*(12+1.5)
a
084 285.900
a/4
021 071
12^12
008 916 100 448 256
quit
$
```

Bei der Ausgabe sehr großer Zahlen wird nach 70 Zeichen eine neue Zeile begonnen; dabei wird am Ende der vorherigen Zeile das Fortsetzungszeichen \ ausgegeben.

Während bei der Ausgabebasis 10 die Ausgabe sehr schnell erfolgt, kann bei anderen Ausgabebasen die Ausgabe von sehr großen Zahlen doch etwas Zeit in Anspruch nehmen. So kann z.B. die Ausgabe einer 500-stelligen Zahl im Dualsystem einige Sekunden dauern.

```
$ bc
24567894^57
1782263814550173585683277686740124485616306516623476715295620871953178\
5701670077302538709108007367738230775521684470572087754474797232360663\
8252061128620275351443423093374524315952545479409512479468064705526965\
8775210683315092773284213608444323029366416073690733344396020504766107\
6777749354928862530311062131095097961218681870085585959494735311715964\
1560124408935784815572191515325047522654994294474054384602265881820528\
64
ibase=2
4.1^4^5
1011001001100011100110110101000100111010101010010011010110101100111011\
1100100101000110100111101100110011100111100000000011100011100010000000\
0000101100010000011111101110100011100101101111000100011011010110101010\
000111111110101110011011110010000101001110111111100101000010001100000\
1101010001110101101011010101000001101111111111111010110000011100101101\
1101111101100110010000100111001001000111100011111010011010011111000011\
0011100111110011011011001000100001101100110010011010111100101101000001\
11001101011100100111010100000001010000001011100100101010000110000110\
00101000000111101011101100111010101110011101110111011110111000001111\
1111000011101011001001011110001101011101011101001111100000110010111000\
001011110100011001001101100110111100110110010111001011100000111111001\
```



```
01010111110100100111001101001011111110110011101101011011111101000\
0001000000000111010001000011100010100110010001110000011010111101100001\
110111101111011011001110111001101010001101011001101000011001011010\
110110111111001000000001011001011000001100010101111101100011000010000\
001110001110110100000011100101010100000101101111100111000111111100101\
0010100010010001110000100100000011100001110001100111000001100010100101\
10111000000100111001110000001100001110011001111100010100110001001101\
101111011100111001111110001110010111111011100011001001010011101010001\
10010100000100110001111011110011011101101101000101010110110110000110\
11111010101011110010100010010000111011110010011101010100001001000100\
1101100101101100000111000111101010000010001101001010011100110110101010\
11100001100000100010101111001110010100110000101000011000011010010000\
011110001010100011110011100101100100000010011101000010110001001010100\
010101011101011110010000110110001010010001111000111101110010010101111\
01000000011110000011110101000110011100111111011111010111001110101111\
1100110000010110111000011101000000000011001010011000100111100000010010\
00100110010010010000101000011011001101100001101010011000011101010001\
011111010111011110010100010010011000100110111100111010110000101010000\
1001001000010100010010010011100101011000010000111110011.010
quit
$
```

Da intern die Berechnungen immer im Zehnersystem (pro Ziffer ein Byte) durchgeführt werden, ist folgendes zu beachten:

- ▶ Die Werte von **ib** **a** und **ob** **a** haben keinerlei Auswirkungen auf die internen Berechnungen, sondern lediglich auf die Ein- und Ausgabe.
- ▶ Der Wert von **scale** bezieht sich immer auf die Nachkommastellen von Zahlen im Zehnersystem, selbst wenn **ib** **a** oder **ob** **a** nicht 10 gesetzt sind. Dies kann dazu führen, daß im entsprechenden Zahlensystem Zahlen mit mehr oder weniger Nachkommastellen ausgegeben bzw. nach der Eingabe gespeichert werden, als mit **scale** festgelegt wurde.

```
$ bc
scale=5
a=1/3
a
0.33333
obase=2
a
.01010101010101
scale=10
obase=10
a=1/4096
a
.0002441406
obase=16 [Ausgabebasis nun 16]
a
.000FFFFF
quit
$
```

9.2.6 A u s g e b e n v o n T e x t

Um einen Text auszugeben, muß dieser mit "." geklammert werden:

```
$ bc
"2 hoch 3 = "
2 hoch 3 = 2^3
8
quit
$
```

9.2.7 F u n k t i o n e n

b läßt die Definition von Funktionen zu, die dann an späterer Stelle aufgerufen werden können. Als Name für eine Funktion muß ein Kleinbuchstabe¹ gewählt werden. Funktionen dürfen gleiche Namen wie Variablen besitzen. Somit können zusätzlich zu den 26 möglichen Variablen noch 26 verschiedene Funktionen definiert werden.

Es ist folgende Syntax für Funktionsdefinitionen² einzuhalten:

```
d e f funktionsname([parameterliste]) {
    [a u t o var1, var2, ..]
    anweisung1
    anweisung2
    :
    :
}
```

Nachfolgend wird eine Funktion mit Namen *m* und zwei formalen Parameter *x* und *y* definiert. Mit der **a u t o**-Zeile wird eine funktionslokale Variable *z* vereinbart, der dann im Funktionskörper das Ergebnis von *x*y* zugewiesen wird, bevor dann dieser Wert von *z* mit der **r e t u r n**-Anweisung zurückgegeben wird.

Ein Aufruf einer Funktion erfolgt mit der Angabe von *funktionsname*, gefolgt von einer geklammerten Liste von aktuellen Parametern, die mit Komma voneinander zu trennen sind:

```
$ bc
define m(x,y) {
    auto z
    z=x*y
    return(z)
}
m(4,7)
28
a=m(4.2,2^4)
```

-
1. Keine Umlaute und kein ß.
 2. [] bedeutet dabei, daß die entsprechende Angabe optional ist.

```

a ↵
67.2
a=m(4.2,m(4,3)) ↵
a ↵
50.4
quit ↵
$

```

Eine Funktion wird beendet, wenn bei ihrer Ausführung entweder das Ende der Funktion erreicht, oder eine **return**-Anweisung ausgeführt wird. Die **return**-Anweisung kann auf zwei verschiedene Arten aufgerufen werden:

return (Rückgabewert: 0)

return (*a*) (Rückgabewert: Wert des angegebenen Ausdrucks *a*)

Funktionslokale Variablen können mit der **auto**-Anweisung vereinbart werden. Eine **auto**-Anweisung muß dabei die erste Anweisung in einer Funktionsdefinition sein. Solche funktionslokalen Variablen werden automatisch bei Eintritt in die Funktion auf 0 gesetzt, und ihre Lebensdauer ist nur auf die Ausführungsdauer der Funktion begrenzt. Das heißt, daß solche Variablen nach einer Funktionsausführung nicht mehr zur Verfügung stehen.

Funktionslokale Variablen können gleiche Namen wie globale Variablen besitzen, die außerhalb der Funktion verwendet werden. In diesem Fall handelt es sich dann trotz des gleichen Namens um zwei verschiedene Variablen. Die Funktions-Parameter werden genauso wie **auto**-Variablen behandelt; der einzige Unterschied ist, daß sie bei Funktionseintritt nicht mit 0, sondern mit dem Wert des entsprechenden aktuellen Parameter initialisiert werden.

```

$ bc ↵
define k(r) { ↵
    auto p, u, f ↵
    p = 3.1415 ↵
    u = 2*r*p ↵
    f = r^2*p ↵
    u ↵
    f ↵
} ↵
x=7.2 ↵
k(x) ↵
45.2376
162.7297
0 [Rückgabewert der Funktion]
u=4.9 ↵
k(u) ↵
30.7867
75.3960
0 [Rückgabewert der Funktion]
u ↵

```

4.9 [Wert des globalen u wurde durch Funktionsaufruf nicht verändert]

[Strg-D]
\$

Funktionen können auch rekursiv aufgerufen werden¹:

```
$ bc
define f(x) {
    if (x > 0) return(x*f(x-1))
    return(1)
}
f(5)
120
f(10)
3628800
f(0)
1
[Strg-D]
$
```

[berechnet Fakultät von x]

Funktionen können auch ohne Parameter definiert und aufgerufen werden:

```
$ bc
define s() {
    p=3.1415
    m=100000
}
s()
0
m
100000
p
3.1415
quit
$
```

[Da p und m nicht mit auto als funktionslokal deklariert wurden,]
[beziehen sich diese Zuweisungen auf die globalen Variablen p und m]

[Rückgabewert der Funktion]

Wenn Funktionen mit einer falschen Parameterzahl aufgerufen werden, kann dies zu unvorhersehbaren Ergebnissen führen:

```
$ bc
define a(x,y) {
    return(x+y)
}
z=a()
save: args
```

1. Im nachfolgenden Beispiel wird bereits von der **if**-Anweisung Gebrauch gemacht. Obwohl diese erst an späterer Stelle besprochen wird, sollte sie hier jedoch keine Verständnisprobleme bereiten, da sie weitgehend der **if**-Anweisung in C entspricht.

```
171 is unimplemented
save: args
stack empty
L?
L?
save:args
z[←]
0
z=a(2,3,4)[←]
z[←]
7
[Strg-D]
$
```

9.2.8 Arrays

In **bc** müssen Arrays nicht wie in höheren Programmiersprachen explizit deklariert werden, sondern werden automatisch durch Zugriff auf ein Array-Element angelegt. Der Zugriff auf ein Array-Element erfolgt durch Angabe des Arraynamens (Kleinbuchstabe¹), gefolgt von **[ausdruck]**. Der Wert von *ausdruck* legt dabei den Array-Index (auch Subscript genannt) fest. Der Index muß immer im Bereich 0..2047 liegen:

```
$ bc[←]
a[1]=1[←]
a[2]=2[←]
a[4]=16[←]
a[3]=9[←]
a[2][←]
2
a[4][←]
16
a[0][←]
00000
[Strg-D]
$
```

Aus dem Beispiel wird deutlich, daß nicht explizit gesetzte Array-Elemente automatisch mit Wert 0 initialisiert sind.

In **bc** sind nur eindimensionale Arrays erlaubt. Arrays können gleiche Namen wie Variablen oder Funktionen besitzen, ohne daß dies zu Problemen führt.

Ist der Wert eines angegebenen Index eine reelle Zahl, wird deren Nachkommateil abgeschnitten, bevor dann der Zugriff auf das entsprechende Array-Element erfolgt.

1. Keine Umlaute und kein ß.

```
$ bc
a=4
a[5]=15.23
a
4
a[5]
15.23
a[2.1]=30
a[2.9]
30
a[2]
30
[Strg-D]
$
```

Array-Elemente können nicht nur in Ausdrücken, sondern auch als aktuelle Parameter bei Funktionsaufrufen oder in **return**-Anweisungen angegeben werden.

```
$ bc
define a(x,y) {
    return(x+y)
}
a[1]=12.3
a[2]=-5
a[3]=14
a(a[1],a[3])
26.3
quit
$
```

Ein Arrayname kann als Parameter bei einer Funktionsdefinition angegeben werden oder mit **auto** deklariert werden. Dabei ist der Arrayname gefolgt von einem leeren Klammernpaar `[]` anzugeben. Wird ein Array als Parameter angegeben, wird beim Aufruf dieser Funktion der Inhalt des als aktueller Parameter angegebenen Arrays vollständig für diese Funktion kopiert. In beiden Fällen handelt es sich dann um funktionslokale Arrays.

```
$ bc
define m(a[],i,j) {
    s=a[i]+a[j]
    return(s)
}
b[1]=10
b[2]=20
b[3]=30
b[4]=40
x=m(b[],2,3)
x
50
define n(i,j) {
```

```

b[1]=1
b[2]=2
b[3]=3
b[4]=4
b[1]=b[i]+b[j]
return(b[1])
}
x=n(2,3)
x
5
b[1]
5 [Inhalt des globalen Arrays b durch Funktionsaufruf
verändert]

define o(i,j) {
  auto b[]
  b[1]=100
  b[2]=200
  b[3]=300
  b[4]=400
  b[1]=b[i]+b[j]
  return(b[1])
}
x=n(2,3)
x
500
b[1]
5 [Inhalt des globalen Arrays b durch Funktionsaufruf nicht
verändert]

quit
$

```

9.2.9 Kommentare

b erlaubt auch die Angabe von Kommentaren. Kommentare müssen dabei wie in C mit `/ ** /` geklammert sein.

9.2.10 if-, while- und for-Anweisung

Diese Anweisungen entsprechen weitgehend den entsprechenden C-Anweisungen:

if <i>b(dingung)</i> anweisung	oder
if <i>b(dingung)</i> { anweisung ₁ ; anweisung ₂ ; ... }	
while <i>b(dingung)</i> anweisung	oder
while <i>b(dingung)</i> { anweisung ₁ ; anweisung ₂ ; ... }	
for <i>ru(druck₁; bedingung; ausdruck₂)</i> anweisung	oder
for <i>ru(druck₁; bedingung; ausdruck₂)</i> { anweisung ₁ ; anweisung ₂ ; ... }	

Eine Bedingung kann mit einem der folgenden Vergleichsoperatoren gebildet werden¹:

$a < b$ a kleiner b
 $a > b$ a größer b
 $a < \leq b$ a kleiner als oder gleich b
 $a > \leq b$ a größer als oder gleich b
 $a = \leq b$ a gleich b
 $a \neq b$ a ungleich b

```
$ cat math.bc ↵
/*---- Fakultaet ----*/
define f(n) {
    auto i,f
    f=1
    for (i=1 ; i<=n ; i++) f=f*i
    return(f)
}

/*---- Binomial-Koeffizient ----*/
define b(m,n) {
    auto x
    x=1
    x=f(m)/(f(n)*f(m-n))
    return(x)
}

/*---- Primzahlen bis n (nach Sieb des Eratosthenes) ----*/
define p(n) {
    auto p[], i, j
    if (n>2047) {
        "zu grosses n (maximal 2047)"
        return(0)
    }
    for (i=2 ; i<=n ; i++) p[i]=i
    for (i=2 ; i<=sqrt(n) ; i++) {
        if (p[i] != 0) {
            for (j=2*i ; j<=n ; j=j+i) p[j]=0
        }
    }
    for (i=2 ; i<=n ; i++) if (p[i] != 0) p[i]
}
$ bc math.bc ↵
f(0) ↵
```

1. a und b stehen dabei für beliebige Ausdrücke.


```
1
f(20) ↵
2432902008176640000
f(5) ↵
120
b(49,6) ↵
13983816
b(10,2) ↵
45
x=p(9999) ↵
zu grosses n (maximal 2047)x=p(60) ↵
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
quit ↵
$
```

9.2.11 Bibliotheksfunktionen

Mit dem Aufruf

bc -l

werden aus einer mathematischen Bibliothek (wie z.B. */usr/lib/libmath.b* oder */lib/libmath.b* oder */usr/local/lib/libmath.b*) bestimmte Funktionen geladen, die dann aufgerufen werden können. Folgende Funktionen werden dabei zur Verfügung gestellt:

- s**  Sinus von x
- c**  Cosinus von x
- a**  Arcustangens von x
- l**  natürlicher Logarithmus von x
- e**  e^x
- j**  Bessel-Funktion

Durch das Laden der Bibliotheksfunktionen wird `scale` auf 20 gesetzt. Neben den hier aufgezählten Funktionen werden oft (abhängig vom jeweiligen System) noch weitere nützliche mathematische Funktionen geladen.

```
$ bc -l
for (i=1 ; i<=10 ; i++) e(i)
2.71828182845904523536
7.38905609893065022723
20.08553692318766774092
54.59815003314423907811
148.41315910257660342111
403.42879349273512260838
1096.63315842845859926372
2980.95798704172827474359
8103.08392757538400770999
22026.46579480671651695790
l(100)
4.60517018598809136803
s(2)+c(3)
-.08069506977476376188
quit
$
```

9.2.12 Zusammengesetzte Zuweisungsop

Ähnlich wie in C können Zuweisungen wie

```
x = x + y
```

zu

```
x = + y
```

zusammengefaßt werden. Allerdings handelt es sich dabei um die alte C-Schreibweise (`= +`), die im neuen C in anderer Reihenfolge (`+ =`) anzugeben ist.

b kennt folgende zusammengesetzte Zuweisungsoperatoren:

```
x = +y    entspricht: x = x + y
```

```
x = -y    entspricht: x = x - y
```

```
x = *y    entspricht: x = x * y
```

```
x = /y    entspricht: x = x / y
```

```
x = %y    entspricht: x = x % y
```

```
x = ^y    entspricht: x = x ^ y
```

In manchen dieser Konstruktionen sind Leerzeichen von Wichtigkeit. So gilt z.B.

```
x = - y    entspricht: x = x - y
```

```
x = -y    entspricht: x = (-y) (wird der negative Wert von y zugewiesen)
```

9.2.13 Weitere Hinweise

- Üblicherweise wird in einer Zeile nur eine Anweisung angegeben. Allerdings ist es erlaubt, mehrere Anweisungen mit Semikolon getrennt in einer Zeile anzugeben.
- Wird eine Zuweisung geklammert angegeben, wird diese ganze Anweisung als Ausdruck interpretiert. So weist z.B.

`(a=13-5)`

nicht nur `a` den Wert 8 zu, sondern gibt diesen Wert auch wie bei einem einfachen Ausdruck aus.

- Wird ein Array-Index in Form einer Zuweisung angegeben, wird auch diese Zuweisung als Ausdruck interpretiert.

```
$ bc
define f(n) {
    auto i
    i=0
    while (i<=n) a[i+1]=i*i
}
f(10)
0
a[2]
1
a[10]
81
a[5]
16
quit
$
```

- Mehrfachzuweisungen sind erlaubt:

`x = y` ist identisch zu `x = ( y = z )`

Es wird also zuerst der Wert von `z` der Variablen `y` zugewiesen und dieser Wert dann der Variablen `x` zugewiesen.

10 Dialogboxen unter Linux

*Unterschieden ist nicht das Schöne vom Guten, das Schöne
ist nur das Gute, das sich lieblich verschleiert uns zeigt.*

Goethe

Unter Linux können mit dem Kommando **dialog** farbige Dialogboxen dargestellt werden. Das Kommando **dialog** wird dort zum Beispiel vom Installationsprogramm **setup** verwendet, um Informationen auszugeben, dem Benutzer bestimmte Optionen auswählen oder ihm Texte (wie z.B. Directory- oder Rechnernamen) eingeben zu lassen. In diesem Kapitel wird eine kurze Beschreibung des **dialog**-Kommandos anhand von Beispielen gegeben.

10.1 Aufrufsyntax des dialog-Kommandos

Das **dialog**-Kommando kann auf drei verschiedene Arten aufgerufen werden:

dialog --clear

löscht den aktuellen Bildschirm, wobei der ganze Bildschirm-Hintergrund auf Blau gesetzt wird.

dialog [--title text] [--clear] boxtyp

Hiermit wird eine entsprechende Dialogbox (*boxtyp*) am Bildschirm erzeugt. Die verschiedenen Arten von Dialogboxen sind nachfolgend anhand von Beispielen beschrieben. Ist auch noch **--clear** angegeben, wird die entsprechende Dialogbox nach Abschluß des Dialogs automatisch wieder gelöscht. Bei dem Dialogboxen-Typ **--infobox** macht ein **--clear** wenig Sinn, da dort nicht auf eine Eingabe des Benutzers gewartet wird und somit die angezeigte Dialogbox sofort wieder gelöscht würde. Da die Angabe der Option **--clear** den Bildschirm in Blau hinterläßt, sollte nach der letzten Dialogbox noch

setterm -default

setterm -clear

aufgerufen werden, um den Bildschirm wieder in den Zustand zu versetzen, der vor der Benutzung von Dialogboxen vorlag. Mit dem Kommando **setterm** kann man im übrigen sehr leicht eigene Bildschirmattribute festlegen. So würde z.B. der Aufruf

setterm -foreground white -background blue -store

den Bildschirm-Hintergrund auf blau und den Vordergrund (Schrift) auf Weiss festlegen. Gibt man dies z.B. in *.profile* an, so werden bereits zu Beginn einer Sitzung entsprechende Bildschirmattribute festgelegt, die dann solange gelten, bis sie explizit geändert werden. Mehr Information zu **setterm** kann in der Manpage zu **setterm** nachgelesen werden.

Zwei weitere mögliche Optionen neben **--clear**, die bei allen Dialogbox-Varianten erlaubt sind, sind

--title *text*

In diesem Fall wird der angegebene Dialogtitel *text* in der obersten Zeile des Dialogfensters angezeigt.

--backtitle *text*

In diesem Fall wird der angegebene Text *text* in der obersten Zeile des Bildschirms angezeigt.

dialog --create-rc *datei*

erstellt eine Konfigurationsdatei für das **dialog**-Kommando.

10.2 Ausgeben von Informationen (**--msgbox**, **--infobox**, **--textbox**)

Die Ausgabe von Informationen kann auf drei verschiedene Arten erfolgen. Bei allen diesen Ausgaben legt das Argument *höhe* die Zeilenzahl und das Argument *breite* die Spaltenzahl der auszugebenden Dialogbox fest.

dialog --msgbox *text höhe breite*

erzeugt eine entsprechende Dialogbox, in der der angegebene *text* angezeigt wird. Der Benutzer muß diese Ausgabe mit *Ok* (durch Eingabe von *Return*) bestätigen, bevor das entsprechende Skript seine Ausführung fortsetzt.

dialog --infobox *text höhe breite*

erzeugt wie **--msgbox** eine entsprechende Dialogbox, in der der angegebene *text* angezeigt wird. Anders als bei **--msgbox** wartet hier **dialog** nicht auf eine Bestätigung durch den Benutzer. **--infobox** wird dazu verwendet, um den Benutzer vor dem Beginn einer zeitaufwendigen Aktion darüber zu informieren, was das Skript gerade bearbeitet, und daß er nun etwas warten muß.

dialog --textbox *datei höhe breite*

erzeugt eine entsprechende Dialogbox, in der der Inhalt der angegebenen Textdatei *datei* angezeigt wird. Ist die Textdatei *datei* größer als die Dialogbox, kann sich der Benutzer mit den Cursor-, Bild- und Positions-Tasten durch den Text bewegen oder in ihm blättern. Bei überlangen Zeilen ist ein Blättern nach rechts und links mit den *RECHTS*- und *LINKS*-Pfeiltasten möglich. Das Vorwärts- und

Rückwärtsblättern ist nicht nur mit den Bild-Tasten, sondern auch mit Leerzeichen (Vorwärts) und Taste **b** (Rückwärts) möglich. **--textbox** wird meist dazu verwendet, um dem Benutzer umfangreichere Erklärungen oder Hilfstexte auszugeben.

Bei der Ausgabe von Dialogboxen wird der angegebene Text automatisch entsprechend der Größe der Dialogbox umgebrochen. Mit **\n** können aber auch explizit Zeilenumbrüche festgelegt werden.

Soll sich ein **dialog**-Kommando über mehrere Zeilen erstrecken, muß am jeweiligen Zeilenende immer das Fortsetzungszeichen **** angegeben werden. Ausnahme sind dabei Zeichenketten, die in Apostrophen ('...') oder Anführungszeichen ("...") angegeben sind. Diese können sich auch ohne dem Fortsetzungszeichen **** über mehrere Zeilen erstrecken.

Das folgende Shellskript *dialinfo* demonstriert die Verwendung der oben beschriebenen **dialog**-Varianten.

```
#!/bin/sh
#=====
# dialinfo: Ausgeben von Informationen
#=====

#-----
# msgbox: Weiter mit "ok"
#-----
dialog --clear --msgbox \
    "Heute ist $(date +%d.%m.%Y)\n\nEs ist $(date +%R) Uhr" 7 25

#-----
# infobox: Skript wird sofort fortgesetzt; deswegen sleep.
#-----
dialog --title "Bitte Warten" --infobox \
    "\nIn 10 Sekunden geht's weiter.\n
    Dann wird der Inhalt der Datei
    dialinfo angezeigt." 7 35
sleep 10

#-----
# textbox: Inhalt dieser Datei dialinfo wird ausgegeben
#-----
dialog --clear --textbox dialinfo 20 75

#-----
# alten Hintergrund wiederherstellen
#-----
setterm -default
setterm -clear
```

Startet man dieses Skript *dialinfo*, gibt es zunächst die folgende Dialogbox aus:



Nachdem der Benutzer mit *Return* sein *Ok* gegeben hat, wird die folgende Dialogbox für 10 Sekunden angezeigt:



Nach dem Ablauf von 10 Sekunden wird in einer neuen Dialogbox der Inhalt des Skripts selbst angezeigt:



In diesem Text kann nun zeilen- oder seitenweise geblättert werden. Drückt der Benutzer z.B. die *Bild-Unten*-Taste oder Leerzeichen, wird ihm der restliche Inhalt der Datei *dialinfo* angezeigt:



```
"AnIn 10 Sekunden geht's weiter.\nDann wird der Inhalt der Datei\ndialinfo angezeigt." 1 35\nsleep 10\n\n#-----\n# textbox: Inhalt dieser Datei dialinfo wird ausgegeben\n#-----\ndialog --clear --textbox dialinfo 20 75\n\n#-----\n# alten Hintergrund wiederherstellen\n#-----\nsetterm -default\nsetterm -clear\n\n< EXIT >
```

Neben Vorwärts- und Rückwärtsblättern ist auch mit den Tasten *Pos1* ein Positionieren auf den Anfang bzw. mit der Taste *Ende* auf das Ende der Datei möglich. Auch ist z.B. mit */* ein Suchen wie bei **more** möglich.

Nach der Eingabe von *Return* durch den Benutzer wird obige Dialogbox verlassen und das Skript *dialinfo* beendet.

10.3 Ja/Nein-Auswahl (--yesno)

Mit der Angabe

dialog --yesno *text höhe breite*

wird eine Dialogbox mit entsprechender *höhe* und *breite* erzeugt, in der sich der angegebene *text* befindet. Anders als bei den vorherigen Dialogboxen muß der Benutzer hier eine *Ja / Nein*-Auswahl treffen. Zwischen *Ja*(Yes) und *Nein*(No) kann dabei der Benutzer mit der *Tab*-Taste und den *Pfeil*-Tasten hin- und herwechseln. Hat er die geeignete Auswahl eingestellt, muß er dann *Return* drücken. Mit der Eingabe der Anfangsbuchstaben **Y** oder **N** ist auch eine direkte Auswahl ohne *Return* möglich.

Das Ergebnis der Benutzer-Entscheidung wird als exit-Status von **dialog** geliefert. Es befindet sich also nach dem **dialog**-Aufruf in der Shellvariablen **\$?**. Der Wert **0** zeigt dabei eine *Ja*- und der Wert **1** eine *Nein*-Entscheidung an. Hat der Benutzer mit *ESC* die Eingabe abgebrochen, so hat **\$?** den Wert **255**.

Das folgende Shellskript *dialyesno* demonstriert die Verwendung von Ja / Nein-Dialogfenstern:

```
#!/bin/sh
#=====
# dialyesno: JA/NEIN-Auswahl
#=====
#-----
# yesno: Frage nach dem Wetter
#-----
dialog --clear --title "Befragung zum Wetter" \
--yesno "\nIst heute schoenes Wetter?" 7 35
auswahl=$?

#-----
# Auswerten der Benutzer-Entscheidung
#-----
if [ $auswahl = 0 ]; then
    dialog --clear --msgbox "Heute ist also schoenes Wetter" 7 50
elif [ $auswahl = 1 ]; then
    dialog --clear --msgbox "Heute ist also schlechtes Wetter" 7 50
else
    dialog --clear --msgbox "Die Wetter-Frage hast Du mit ESC abgebrochen"
7 30
fi
#-----
# alten Hintergrund wiederherstellen
#-----
setterm -default
setterm -clear
```

Startet man dieses Skript *dialyesno*, gibt es zunächst die folgende Dialogbox aus:



Wählt der Benutzer *Ja* (Yes), erscheint als nächstes das folgende Fenster:



Hätte der Benutzer *Nein* (No) gewählt, wäre das folgende Fenster erschienen:



Hätte der Benutzer bei der ersten Dialogbox *ESC* gedrückt, wäre ihm folgendes Fenster angezeigt worden:



10.4 Eingabe von Strings (--inputbox)

Mit der Angabe

dialog --inputbox *text höhe breite*

wird eine Dialogbox mit entsprechender *höhe* und *breite* erzeugt, in der sich der angegebene *text* befindet. Nun kann der Benutzer einen String eingeben. Der eingegebene String wird auf die Standardfehlerausgabe geschrieben. Es empfiehlt sich hier also die Standardfehlerausgabe mit **2>** in eine temporäre Datei umzulenken, um dann nach Verlassen der Dialogbox den eingegebenen Text in dieser temporären Datei verfügbar zu haben.

Das folgende Shellskript *dialinput* zeigt hierzu ein Beispiel.

```
#!/bin/sh
#=====
# dialinput: Eingabe von Text
#=====

#-----
# inputbox: Frage nach dem Namen
#-----
dialog --clear --title "Namenseingabe" \
--inputbox "Wie heisst Du denn?" 7 40 2>/tmp/dialinput.$$
antwort=$?

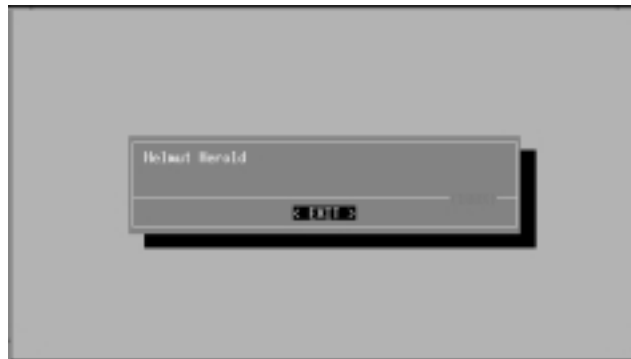
#-----
# Ausgeben der Benutzer-Eingabe
#-----
if [ $antwort = 0 ]; then      # wurde Dialog mit Ok beendet?
    dialog --clear --textbox /tmp/dialinput.$$ 7 50
    name=$(cat /tmp/dialinput.$$)
    dialog --clear --msgbox "Dein Name ist also '$name'" 7 50
elif [ $antwort = 1 ]; then
    dialog --clear --msgbox "Du willst also anonym bleiben (Cancel)" 7 50
else
    dialog --clear --msgbox "Du willst also anonym bleiben (ESC)" 7 50
fi
rm /tmp/dialinput.$$

#-----
# alten Hintergrund wiederherstellen
#-----
setterm -default
setterm -clear
```

Startet man dieses Skript *dialinput*, gibt es zunächst die folgende Dialogbox aus:



Nachdem der Benutzer seinen Namen eingegeben und diesen mit *Return* quittiert hat, wird ihm sein eingegebener Name zunächst in einer Textbox ausgegeben



und danach nochmals in einem *msgbox*-Dialogfenster angezeigt:



Ein Wechseln zwischen Texteingabe, *Ok* und *Cancel* ist mit der *Tab*-Taste oder auch mit der *Pfeil-oben*- und *Pfeil-unten*-Taste möglich. Hätte sich der Benutzer bei der ersten Dialogbox nicht für *Ok*, sondern für *Cancel* entschieden, wäre ihm folgendes Fenster angezeigt worden:



Hätte der Benutzer bei der ersten Dialogbox *ESC* gedrückt, wäre ihm folgendes Fenster angezeigt worden:



10.5 Menüauswahl (--menu)

Mit der Angabe

dialog --menu *text höhe breite menühöhe [kennung menütext]*

wird eine Dialogbox mit entsprechender *höhe* und *breite* erzeugt, in der sich an oberster Stelle der angegebene *text* befindet. Zusätzlich muß hier noch über *menühöhe* die Anzahl der anzuzeigenden Menüzeilen angegeben werden. Die nächsten Angaben-Paare legen die einzelnen Menüpunkte fest.

kennung gibt dabei eine Kurzauswahl an. Hierfür sollten Buchstaben, Zahlen oder andere Zeichen angegeben werden. Es sind jedoch auch Strings wie z.B. Datei- oder Kommandonamen erlaubt. Zur schnellen Auswahl eines Menüpunktes kann der Benutzer das erste Zeichen von *kennung* eingeben.

Der zweite String *menütext* ist eine Kurzbeschreibung des jeweiligen Menüpunktes.

Zur Auswahl eines Menüpunktes stehen dem Benutzer mehrere Möglichkeiten zur Verfügung:

1. Mit *Pfeil-oben*- oder *Pfeil-unten*-Taste bzw. mit den Tasten - oder + auf entsprechenden Menüpunkt bewegen.
2. Durch Eingabe des ersten Zeichens von *kennung* direkt auf entsprechenden Menüpunkt positionieren.
3. Durch Eingabe einer Ziffer direkt auf den *n*.ten Menüpunkt in der sichtbaren Menüliste positionieren.

Wenn eine Menüliste mehr als die über *menühöhe* festgelegten Zeilen besitzt, wird immer nur ein entsprechender Ausschnitt aus der Menüliste angezeigt. In diesem Fall ist ein Vorwärtsblättern im Menü mit der Taste + oder *Pfeil-unten*-Taste und ein Zurückblättern mit der Taste - oder *Pfeil-oben*-Taste möglich.

Hat der Benutzer seinen entsprechenden Menüpunkt ausgewählt, muß er diesen mit der Eingabe von *Return* bei *Ok* quittieren. Die *kennung* des ausgewählten Menüpunktes wird dann im Verlassen der Menü-Dialogbox auf die Standardfehlerausgabe geschrieben. Es empfiehlt sich hier also wieder die Standardfehlerausgabe mit 2> in eine temporäre Datei umzulenken.

Das folgende Shellskript *dialmenu* ist ein Demonstrationsbeispiel zur Menüauswahl beim **dialog**-Kommando:

```
#!/bin/sh
#=====
# dialmenu: Menue-Auswahl
#=====

#-----
# menu: Auswahl eines Menuepunktes
#-----
dialog --clear --title "Menueauswahl" \
      --menu "Was soll ich denn nun tun?" 12 40 4 \
          "D" "Drucken" \
          "S" "Suchen" \
          "E" "Ersetzen" \
          "N" "Numerieren" 2>/tmp/dialmenu.$$
auswahl=$?

#-----
# Ausgeben des vom Benutzer gewaehlten Menuepunktes
#-----
if [ $auswahl = 0 ]; then      # wurde Dialog mit Ok beendet?
    menuepkt=$(cat /tmp/dialmenu.$$)
    dialog --clear --msgbox "Du hast den Menuepunkt $menuepkt gewaehlt" 7 50
elif [ $auswahl = 1 ]; then
    dialog --clear --msgbox "Du hast keinen Menuepunkt gewaehlt (Cancel)" 7 50
else
    dialog --clear --msgbox "Du hast keinen Menuepunkt gewaehlt (ESC)" 7 50
fi
rm /tmp/dialmenu.$$

#-----
# alten Hintergrund wiederherstellen
#-----
setterm -default
setterm -clear
```

Startet man dieses Skript *dialmenu*, gibt es zunächst die folgende Dialogbox aus:



Nachdem der Benutzer z.B. den Menüpunkt **E** (*Ersetzen*) ausgewählt und dies mit *Return* quittiert hat, wird ihm seine getroffene Wahl in einer eigenen Dialogbox nochmals angezeigt:



Hätte sich der Benutzer bei der ersten Dialogbox nicht für *Ok*, sondern für *Cancel* entschieden, wäre ihm folgendes Fenster angezeigt worden:



Hätte der Benutzer bei der ersten Dialogbox *ESC* gedrückt, wäre ihm folgendes Fenster angezeigt worden:



10.6 Auswahl von Optionen (--checklist)

Die Auswahl von Optionen ist sehr ähnlich zur Menüauswahl. Mit der Angabe **dialog --checklist** *text höhe breite listenhöhe [kennung optionstext startzustand]*

wird eine Dialogbox mit entsprechender *höhe* und *breite* erzeugt, in der sich an oberster Stelle der angegebene *text* befindet. Über *listenhöhe* muß die Anzahl der anzuzeigenden Optionen festgelegt werden. Die nächsten Dreiergruppen legen die einzelnen Optionen fest.

kennung gibt dabei eine Kurzauswahl an. Hierfür sollten Buchstaben, Zahlen oder andere Zeichen angegeben werden. Es sind jedoch auch Strings erlaubt.

Der zweite String *optionstext* ist eine Kurzbeschreibung der jeweiligen Option.

Für *startzustand* ist entweder **on** oder **off** anzugeben, je nachdem, ob diese Option zu Beginn ein- bzw. ausgeschaltet sein soll.

Zur Anwahl einer Option stehen mehrere Möglichkeiten zur Verfügung:

1. Mit *Pfeil-oben*- oder *Pfeil-unten*-Taste bzw. mit den Tasten - oder +.
2. Durch Eingabe des ersten Zeichens von *kennung* direkt positionieren.
3. Durch Eingabe einer Ziffer direkt auf *n*.te Option in sichtbarer Optionsliste positionieren.

Eingeschaltete Optionen sind immer mit *X* in der Optionsliste gekennzeichnet. Hat der Benutzer auf einer entsprechenden Option positioniert, kann er die *Leerzeichen*-Taste als Schalter zum Ein- bzw. Ausschalten dieser Option benutzen.

Besitzt eine Optionsliste mehr als die über *listenhöhe* festgelegten Zeilen, wird immer nur ein entsprechender Ausschnitt aus der Optionsliste angezeigt. In diesem Fall ist ein Vorwärtsblättern in der Liste mit der Taste + oder *Pfeil-unten*-Taste und ein Zurückblättern mit der Taste - oder *Pfeil-oben*-Taste möglich.

Nachdem der Benutzer die Optionen seinen Wünschen entsprechend gesetzt hat, muß er dies mit der Eingabe von *Return* bei *Ok* quittieren. Die *kennungen* der eingeschalteten Optionen werden dann beim Verlassen der Dialogbox auf die Standardfehlerausgabe geschrieben. Es empfiehlt sich hier also wieder, die Standardfehlerausgabe mit **2>** in eine temporäre Datei umzulenken.

Das folgende Shellskript *dialcheck* ist ein Demonstrationsbeispiel zur Auswahl von Optionen beim **dialog**-Kommando:

```
#!/bin/sh
#=====
# dialcheck: Setzen von ein oder mehreren Optionen
#=====

#-----
# checklist: Setzen von Optionen
#-----
dialog --clear --title "Auswahl von Optionen" \
--checklist "\nSetzen Sie die entspr. Optionen!" 15 40 5 \
    "G" "Nur Grossbuchstaben" off \
    "k" "Nur Kleinbuchstaben" off \
    "B" "Gross- und Kleinbuchstaben" on \
    "N" "Mit Zeilennummern" on \
    "D" "Diagnose-Meldungen" off 2>/tmp/dialcheck.$$
auswahl=$?

#-----
# Ausgeben der vom Benutzer gesetzten Optionen
#-----
if [ $auswahl = 0 ]; then # wurde Dialog mit Ok beendet?
    optionen=$(cat /tmp/dialcheck.$$)
    dialog --clear --msgbox "Folgende Optionen sind gesetzt: $optionen" 7 60
elif [ $auswahl = 1 ]; then
    dialog --clear --msgbox "Setzen der Optionen wurde abgebrochen (Cancel)" 7
55
else
    dialog --clear --msgbox "Setzen der Optionen wurde abgebrochen (ESC)" 7
55
fi
rm /tmp/dialcheck.$$

#-----
# alten Hintergrund wiederherstellen
#-----
setterm -default
setterm -clear
```

Startet man dieses Skript *dialcheck*, gibt es zunächst die folgende Dialogbox aus, bei der die Optionen **B** (Gross- und Kleinbuchstaben) und **N** (Mit Zeilennummern) eingeschaltet sind.



Nachdem der Benutzer z.B. mit der Leertaste die Option **B** ausgeschaltet und die beiden Optionen **G** und **D** eingeschaltet hat, und dies dann mit *Return* quittiert hat, werden ihm seine gesetzten Optionen in einer eigenen Dialogbox nochmals angezeigt:



Hätte sich der Benutzer bei der ersten Dialogbox nicht für *Ok*, sondern für *Cancel* entschieden, wäre ihm folgendes Fenster angezeigt worden:



Hätte der Benutzer bei der ersten Dialogbox *ESC* gedrückt, wäre ihm folgendes Fenster angezeigt worden:



Manchmal ist es notwendig, daß eine Optionsliste erst zur Laufzeit generiert wird. Dazu muß man das entsprechende **dialog**-Kommando während der Laufzeit erst in einer temporären Datei »zusammenbauen«. Zur Ausführung dieses konstruierten **dialog**-Kommandos muß dann diese temporäre Datei gestartet werden.

Das folgende Shellskript *dialcheck2* ist ein Demonstrationsbeispiel für die Auswahl von Optionen zur Laufzeit. Es erzeugt in der temporären Datei */tmp/dialogt.\$\$* ein **dialog**-Kommando, das als Optionsliste alle Dateinamen des Working-Directorys enthält. Diese temporäre Datei startet es dann, um dem Benutzer nur die Dateien des Working-Directories auswählen zu lassen, für die ein **ls -alR** auszuführen ist.

```
#!/bin/sh
#=====
# dialcheck2: Generieren einer Optionsliste zur Laufzeit
#=====

#-----
# Generieren der Optionen in einer temporaeren Datei
#-----
cat >/tmp/dialopt.$$ <<EOF
dialog --clear --title "Auswahl von aufzulistenden Dateien" \
    --checklist '\nWaehle bitte die Dateien aus,
fuer die ein "ls -aR" \
durchgefuehrt werden soll' 20 55 12 \
EOF
for i in $(ls -a)
do
    echo "'$i' ' off \\" >>/tmp/dialopt.$$
done
echo "2>/tmp/dialcheck.$$" >>/tmp/dialopt.$$

#-----
# Benutzer aus den automatisch generierten Optionen waehlen lassen
#-----
sh /tmp/dialopt.$$
auswahl=$?

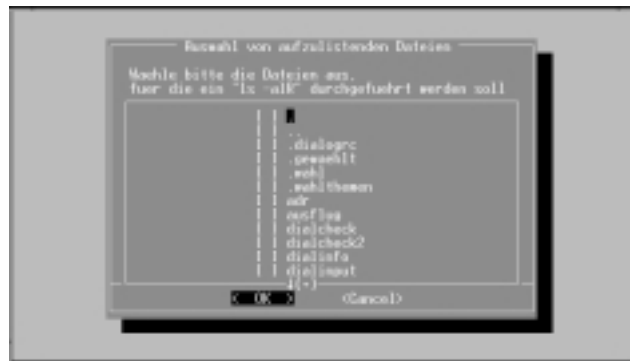
#-----
# Entsprechend der vom Benutzer gesetzten Optionen auswerten
#-----
if [ $auswahl = 0 ]; then    # wurde Dialog mit Ok beendet?
    optionen=$(cat /tmp/dialcheck.$$)
    dateiliste=""
    for i in $optionen
    do
        datei=$(expr $i : '"\(.*)"')
        dateiliste="$dateiliste $datei"
    done
    if [ "$dateiliste" != "" ]; then
        ls -aR $dateiliste >/tmp/dialog.datlist.$$
    else
        echo "keine Dateien ausgewaehlt" >/tmp/dialog.datlist.$$
    fi
    dialog --clear --textbox /tmp/dialog.datlist.$$ 22 70
    rm /tmp/dialog.datlist.$$
elif [ $auswahl = 1 ]; then
    dialog --clear --msgbox "Setzen der Optionen abgebrochen (Cancel)" 7 55
else
    dialog --clear --msgbox "Setzen der Optionen wurde abgebrochen (ESC)" 7 55
fi
rm /tmp/dialcheck.$$
rm /tmp/dialopt.$$
```

```
#-----  
# alten Hintergrund wiederherstellen  
#-----  
setterm -default  
setterm -clear
```

Die von dialcheck2 konstruierte temporäre Datei `/tmp/dialopt.$$` kann z.B. folgenden Inhalt haben:

```
dialog --clear --title "Auswahl von aufzulistenden Dateien" \  
        --checklist '\nWaehle bitte die Dateien aus,  
fuer die ein "ls -alR" durchgefuehrt werden soll' 20 55 12 \  
'.' '' off \  
'..' '' off \  
'dialogrc' '' off \  
'gewaehlt' '' off \  
'wahl' '' off \  
'wahlthemen' '' off \  
'adr' '' off \  
'ausflug' '' off \  
'dialcheck' '' off \  
'dialcheck2' '' off \  
'dialinfo' '' off \  
'dialinput' '' off \  
'dialmenu' '' off \  
'dialyesno' '' off \  
'mittagessen' '' off \  
'screen.dump' '' off \  
'screendump' '' off \  
'silvester' '' off \  
'test' '' off \  
'wahl' '' off \  
'wahl_inf' '' off \  
2>/tmp/dialcheck.98
```

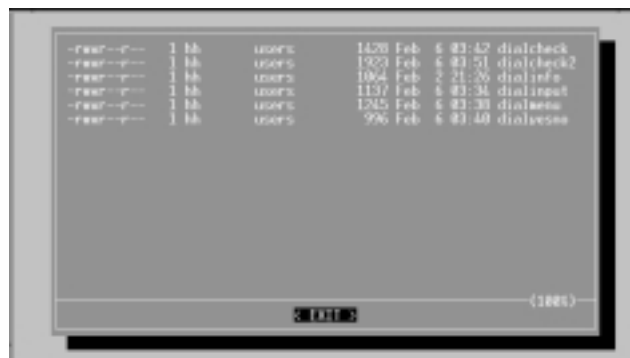
Startet man das Skript **dialcheck2**, gibt es z.B. zunächst die folgende Dialogbox aus:



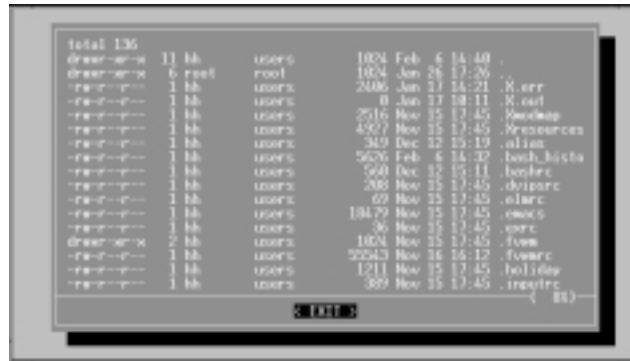
Nachdem der Benutzer aus der Dateiliste z.B. alle Dateien ausgewählt hat, die mit **dial** beginnen, hätte die Dialogbox folgendes Aussehen:



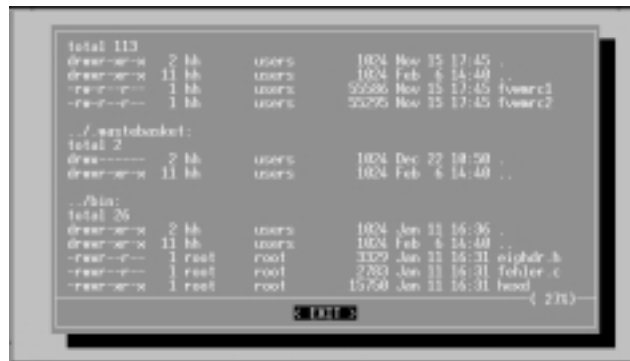
Quittiert der Benutzer nun diese Auswahl mit **Return**, wird für alle Dateien, deren Name mit **dial** beginnt, ein **ls -alR** durchgeführt und in einer eigenen Dialogbox angezeigt:



Hätte der Benutzer in der ersten Dialogbox z.B. .. (Parent-Directory) ausgewählt, wäre ihm z.B. die folgende Dialogbox angezeigt worden:



In dieser kann der Benutzer nun mit / oder ? (wie bei **more**) nach bestimmten Dateinamen suchen oder beliebig vor- und zurückblättern, wie z.B.:



10.7 Auswahl einer von mehreren Alternativen (--radiolist)

Die Auswahl einer von mehreren Alternativen ist sehr ähnlich zur Auswahl von Optionen (siehe vorheriges Kapitel). Mit der Angabe

dialog --radiolist *text höhe breite listenhöhe* [*kennung alternativtext startzustand*]
....

wird eine Dialogbox mit entsprechender *höhe* und *breite* erzeugt, in der sich an oberster Stelle der angegebene *text* befindet. Über *listenhöhe* muß die Anzahl der anzuzeigenden Alternativen festgelegt werden. Die nächsten Dreiergruppen legen die einzelnen Alternativen fest.

kennung gibt dabei eine Kurzauswahl an. Hierfür sollten Buchstaben, Zahlen oder andere Zeichen angegeben werden. Es sind jedoch auch Strings erlaubt.

Der zweite String *alternativtext* ist eine Kurzbeschreibung der jeweiligen Alternative.

Für *startzustand* ist entweder **on** oder **off** anzugeben, je nachdem ob diese Alternative zu Beginn ein- bzw. ausgeschaltet sein soll.

Zur Anwahl einer Alternative stehen mehrere Möglichkeiten zur Verfügung:

1. Mit *Pfeil-oben*- oder *Pfeil-unten*-Taste bzw. mit den Tasten – oder +.
2. Durch Eingabe des ersten Zeichens von *kennung* direkt positionieren.
3. Durch Eingabe einer Ziffer direkt auf *n*.te Alternative in sichtbarer Alternativliste positionieren.

Eine ausgewählte Alternative ist immer mit *X* gekennzeichnet. Hat der Benutzer auf einer entsprechenden Alternative positioniert, kann er mit der *Leerzeichen*-Taste diese auswählen, was bewirkt, daß die bisher ausgewählte Alternative nun nicht mehr aktiv ist, da von allen Alternativen in der Liste immer nur eine aktiv sein kann.

Besitzt eine Alternativliste mehr als die über *listenhöhe* festgelegten Zeilen, so wird immer nur ein entsprechender Ausschnitt aus der Alternativliste angezeigt. In diesem Fall ist ein Vorwärtsblättern in der Liste mit der Taste + oder *Pfeil-unten*-Taste und ein Zurückblättern mit der Taste – oder *Pfeil-oben*-Taste möglich.

Nachdem der Benutzer die gewünschte Alternative ausgewählt hat, muß er dies mit der Eingabe von *Return* bei *Ok* quittieren. Die *kennungen* der eingeschalteten Optionen werden dann beim Verlassen der Dialogbox auf die Standardfehlerausgabe geschrieben. Es empfiehlt sich hier also wieder die Standardfehlerausgabe mit **2>** in eine temporäre Datei umzulenken.

Das folgende Shellskript *dialradio* ist ein Demonstrationsbeispiel zur Auswahl einer Alternative aus mehreren beim **dialog**-Kommando:

```
#!/bin/sh
#=====
# dialradio: Setzen von Radiobuttons (Auswahl einer Alternative)
#=====

#-----
# radiolist: Setzen von Radiobuttons
#-----

dialog --clear --title "Auswahl einer der folgenden Alternative" \
--radiolist "\nWaehlen Sie das fuer Sie zutreffende!" 15 50 4 \
    "k" "Kind" (Alter unter 14)" off \
    "j" "Jugendlicher" (Alter 15-21)" off \
```

```
"m" "Mittleres Alter (Alter 21-60)" on \
"s" "Senior (Alter ueber 60)" off 2>/tmp/dialradio.$$
auswahl=$?

#-----
# Ausgeben der ausgewaehlten Alternative
#-----
if [ $auswahl = 0 ]; then # wurde Dialog mit Ok beendet?
    alternative=$(cat /tmp/dialradio.$$)
    if [ $alternative == 'k' ]; then
        text="ein Kind"
    elif [ $alternative == 'j' ]; then
        text="ein Jugendlicher"
    elif [ $alternative == 'm' ]; then
        text="zwischen 21 und 60 Jahre alt"
    elif [ $alternative == 's' ]; then
        text="schon ueber 60 Jahre"
    fi
    dialog --clear --msgbox "Sie sind also $text" 7 60
elif [ $auswahl = 1 ]; then
    dialog --clear --msgbox "Auswahl wurde abgebrochen (Cancel)" 7 45
else
    dialog --clear --msgbox "Auswahl wurde abgebrochen (ESC)" 7 45
fi
rm /tmp/dialradio.$$

#-----
# alten Hintergrund wiederherstellen
#-----
setterm -default
setterm -clear
```

Startet man dieses Skript *dialradio*, gibt es zunächst die folgende Dialogbox aus, bei der die Alternative **m** (**Mittleres Alter**) ausgewählt ist, was die Voreinstellung ist.



Nachdem der Benutzer z.B. mit der Leertaste die Alternativ j ausgewählt hat, ergibt sich das folgende Aussehen der Dialogbox:



Nachdem der Benutzer diese Auswahl mit *Return* quittiert hat, wird ihm seine getroffene Auswahl in einer eigenen Dialogbox nochmals angezeigt:



10.8 Ausgabe von Prozentbalken (--gauge)

Mit der Angabe

dialog --gauge *text höhe breite prozent*

wird eine Dialogbox mit entsprechender *höhe* und *breite* erzeugt, in der sich der entsprechende *text* befindet. Unten in dieser Box wird ein Prozentbalken angezeigt, in dem die *prozent*-Angabe ausgegeben und der Cursor entsprechend dieser Angabe positioniert wird. Veränderungen in diesem Prozentbalken müssen über die Standardeingabe erfolgen. Liest dieses **dialog**-Kommando von der Standardeingabe eine neue Prozentzahl, paßt es den Prozentbalken entspre-

chend an. Liest dieses Kommando in der Standardeingabe XXX, dann wird der ab hier gelesene Text bis zum abschließenden XXX als neuer Text in der Dialogbox angezeigt. Eine solche **gauge**-Dialogbox wird beendet, wenn das entsprechende **dialog**-Kommando EOF von der Standardeingabe liest.

Das folgende Shellskript *dialgauge* ist ein Demonstrationsbeispiel zur Anzeige von Prozentbalken:

```
#!/bin/sh
#=====
# dialgauge: Ausgeben eines Prozentbalkens
#=====

#-----
# gauge: Ausgeben eines Prozentbalkens
#-----
(sleep 2;
 for i in 10 25 50 75 100
 do
   if [ $i == 50 ]; then
     echo "XXX"
     echo $i
     echo "Die Haelfte haben wir schon\nBald ist es vorbei"
     echo "XXX"
   else
     echo $i
   fi
   sleep 2
 done ) |
dialog --backtitle "Prozentbalken" --gauge "Bitte Warten" 7 35 1

#-----
# alten Hintergrund wiederherstellen
#-----
setterm -default
setterm -clear
```

Startet man dieses Skript *dialgauge*, gibt es zunächst die folgende Dialogbox aus, bei der die Prozentzahl 1 angezeigt wird, was über das **dialog**-Kommando festgelegt wurde.



Nach zwei Sekunden wird die folgende Box (10%) angezeigt:



Nach weiteren zwei Sekunden wird die folgende Box (25%) angezeigt:



Nach weiteren zwei Sekunden wird die folgende Box (50%) angezeigt, wobei hier nun ein neuer Text ausgegeben wird:



Nach weiteren zwei Sekunden wird die folgende Box (75%) angezeigt:



Nach weiteren zwei Sekunden wird schließlich die folgende letzte Box (100%) angezeigt:



10.9 Konfiguration von dialog-Ausgaben

Mit dem Aufruf

dialog --create-rc *datei*

kann man sich eine eigene Konfigurationsdatei für das **dialog**-Kommando erstellen lassen, die man entsprechend ändern kann, um das Aussehen von Dialogboxen seinen eigenen Wünschen anzupassen.

Um die Einstellungen für Dialogboxen zu erfahren, geht das Kommando **dialog** wie folgt vor:

1. Ist die Environment-Variable **DIALOGRC** gesetzt, legt der darin angegebene Name die zu verwendende Konfigurationsdatei fest.
2. Ist **DIALOGRC** nicht gesetzt oder existiert die in **DIALOGRC** angegebene Datei nicht, versucht **dialog** den Inhalt der Datei **\$HOME/.dialogrc** zu lesen.
3. Sollte auch die Datei **\$HOME/.dialogrc** nicht existieren, verwendet **dialog** seine default-Einstellungen.

Nachfolgend ist eine typische Konfigurationsdatei für das Kommando **dialog** gezeigt, die entsprechend editiert werden kann, um die Dialogboxen den eigenen Wünschen entsprechend anzupassen.

```
#
# Run-time configuration file for dialog
#
# Automatically generated by "dialog --create-rc <file>"
#
#
# Types of values:
#
# Number      - <number>
# String      - "string"
# Boolean      - <ON|OFF>
# Attribute   - (foreground,background,highlight?)
#
#
# Shadow dialog boxes? This also turns on color.
use_shadow = ON
#
# Turn color support ON or OFF
use_colors = ON
#
# Screen color
screen_color = (WHITE,BLUE,OFF)
#
# Shadow color
shadow_color = (WHITE,BLACK,OFF)
```

```
# Dialog box color
dialog_color = (BLACK,CYAN,OFF)

# Dialog box title color
title_color = (YELLOW,CYAN,ON)

# Dialog box border color
border_color = (CYAN,CYAN,ON)

# Active button color
button_active_color = (WHITE,BLUE,ON)

# Inactive button color
button_inactive_color = (BLACK,CYAN,OFF)

# Active button key color
button_key_active_color = (WHITE,BLUE,ON)

# Inactive button key color
button_key_inactive_color = (BLACK,CYAN,OFF)

# Active button label color
button_label_active_color = (WHITE,BLUE,ON)

# Inactive button label color
button_label_inactive_color = (BLACK,CYAN,ON)

# Input box color
inputbox_color = (BLUE,WHITE,OFF)

# Input box border color
inputbox_border_color = (CYAN,CYAN,ON)

# Search box color
searchbox_color = (YELLOW,WHITE,ON)

# Search box title color
searchbox_title_color = (WHITE,WHITE,ON)

# Search box border color
searchbox_border_color = (RED,WHITE,OFF)

# File position indicator color
position_indicator_color = (RED,CYAN,OFF)

# Menu box color
menubox_color = (BLACK,CYAN,OFF)

# Menu box border color
menubox_border_color = (CYAN,CYAN,ON)
```



```
# Item color
item_color = (BLACK,CYAN,OFF)

# Selected item color
item_selected_color = (WHITE,BLUE,OFF)

# Tag color
tag_color = (YELLOW,CYAN,ON)

# Selected tag color
tag_selected_color = (WHITE,BLUE,OFF)
# Tag key color
tag_key_color = (RED,CYAN,OFF)

# Selected tag key color
tag_key_selected_color = (RED,BLUE,ON)

# Check box color
check_color = (BLACK,CYAN,OFF)

# Selected check box color
check_selected_color = (WHITE,CYAN,ON)

# Up arrow color
uarrow_color = (GREEN,CYAN,ON)

# Down arrow color
darrow_color = (GREEN,CYAN,ON)
```

10.10 Anwendungsbeispiel

Die beiden nachfolgenden tcsh-Skripts *wahl* und *wahl_info* sollen Umfragen zu beliebigen Themen automatisieren. Diese Aufgabenstellung wurde bereits in Kapitel 6.24, wo sich hierzu auch eine genauere Beschreibung befindet, in Form von csh-Skripts gelöst.

Hier nun wird diese Aufgabenstellung unter zu Hilfenahme des Kommandos **dialog** gelöst, um somit dem Benutzer über eine ansprechendere Oberfläche seine Wahl treffen zu lassen (Skript *wahl*) bzw. sich über den aktuellen Stand der Wahl zu informieren (Skript *wahl_info*).

10.10.1 Das Skript wahl

```
#!/bin/tcsh -f
#
#   wahl - Skript, welches Abstimmungen jeglicher Art automatisiert.
#

setterm -store
set dir = $0
set dir = $dir:h
if ("$dir" == "$0") set dir = "."

set themen_datei = $dir/.wahlthemen

if (-e $themen_datei) then
    # awk-Aufruf bewirkt das Ignorieren von Leerzeilen
    # (siehe auch Folgeband dieser Reihe (Linux-Unix Profitools)
    cat $themen_datei | awk '$0 !~ /^[ \t]*$/ ' >~/themen_tmp.$$
    @ themen_zahl = `cat ~/themen_tmp.$$ | wc -l`
else
    @ themen_zahl = 0
endif

if ($themen_zahl == 0) then
    dialog --msgbox "Es steht zur Zeit kein Thema zur Wahl" 10 45
    setterm -default
    setterm -clear
    rm -f ~/themen_tmp.$$
    exit 1
else
    set menue = `cut -d: -f1 ~/themen_tmp.$$ | awk '{printf("\'%s\'\\n", $0)}' |
nl`
    echo -n '( dialog --clear ' >~/menue.$$
    echo -n '--menu "Folgende Themen stehen zur Wahl" 12 60 ' >>~/menue.$$
    echo -n "$themen_zahl $menue:q " >>~/menue.$$
    echo " >/dev/tty ) >& ~/tmp.$$" >>~/menue.$$
    source ~/menue.$$
    set ergebn = $status
    setterm -default -clear
    set thema_wahl = `cat ~/tmp.$$`
    setterm -default
    setterm -clear
    rm -f ~/tmp.$$
    rm -f ~/menue.$$
endif

if ( $ergebn == 0 ) then
    set thema = `cut -d: -f2 ~/themen_tmp.$$`
    set thema_datei = $thema[$thema_wahl]
    set thema_zeile = "`awk 'NR == $thema_wahl' ~/themen_tmp.$$ | cut -d: -f1`"
else
```

```
rm -f ~/themen_tmp.$$
exit (1)
endif
rm -f ~/themen_tmp.$$

if (! -e $dir/$thema_datei ) then
    dialog --msgbox "Wahldatei --- $dir/$thema_datei --- nicht vorhanden" 8 60
    setterm -default
    setterm -clear
    exit (1)
endif

cd $dir/$thema_datei

if (-e .geschlossen) then
    cat <<EOF >>/tmp.$$
    Wahl fuer $thema_zeile
        bereits abgeschlossen
    Rufen Sie --- wahl_info --- auf, um Wahlergebnisse zu erfahren
EOF
    dialog --textbox ~/tmp.$$ 8 70
    setterm -default
    setterm -clear
    rm -f ~/themen_tmp.$$
    rm -f ~/tmp.$$
    exit (0)
endif

if (-e .geheim) then
    dialog --msgbox "Wahl zu ***$thema_zeile***\n\n          ist eine geheime
Wahl" 8 60
    setterm -default
    setterm -clear
    set ist_geheim = 1
else
    set ist_geheim = 0
endif

if (-e .stimmzahl) then
    set stimm_zahl = (`cat .stimmzahl`)
else
    set stimm_zahl = 1
endif

set waehler = `logname`
touch .gewaehlt
set wieoft_gewaehlt = (`fgrep $waehler .gewaehlt | wc -l `)

if ( $wieoft_gewaehlt >= $stimm_zahl ) then
    dialog --msgbox "Tut mir leid, aber sie haben schon alle Ihre Stimmen
($stimm_zahl) zu\
```

```
\n      ***$thema_zeile*** abgegeben" 9 75
setterm -default
setterm -clear
exit 0
endif

set auswahl = .auswahl
if (-e $auswahl) then
    cat $auswahl | awk '$0 !~ /^[ \t]*$/' >~/auswahl_tmp.$$
    @ auswahl_zahl = `awk '$0 !~ /^[ \t]*$/' ~/auswahl_tmp.$$ | wc -l`
else
    @ auswahl_zahl = 0
endif

if ($auswahl_zahl == 0) then
    dialog --msgbox "Es ist zur Zeit keinerlei Wahl moeglich" 10 45
    setterm -default
    setterm -clear
    rm -f ~/auswahl_tmp.$$
    exit 1
else
    while (1)
        set menue = `cut -d: -f1 ~/auswahl_tmp.$$ | awk '{printf("\'%s'\n",$0)
    }' | nl`
    echo -n '( dialog --clear ' >~/menue.$$
    echo -n '--menu "Folgende Wahlmoeglichkeiten besitzen Sie" 20 60 ' >>~/
menue.$$
    echo -n "$auswahl_zahl $menue:q " >>~/menue.$$
    echo ">/dev/tty ) >& ~/tmp.$$" >>~/menue.$$
    source ~/menue.$$
    set ergebn = $status
    if ($ergebn != 0) then
        dialog --clear --msgbox "Vielen Dank\n\nProgramm $0 beendet sich" 9
70
        break
    endif
    setterm -default -clear
    set wahl = `cat ~/tmp.$$`
    rm -f ~/tmp.$$
    rm -f ~/menue.$$
    set wahl_zeile = "`awk 'NR == $wahl' ~/auswahl_tmp.$$`"
    dialog --clear --yesno "Sie haben ***$wahl_zeile*** gewaehlt\n\
        Wollen Sie diese Stimme nun abgeben?" 8 70
    set ergebn = $status
    if ($ergebn == 0) then
        echo $waehler >>! .gewaehlt
        if ($ist_geheim == 1) then
            echo "`date` | $wahl_zeile" >>! .wahlurne
            dialog --clear --msgbox "Vielen Dank.\
                Ihre Stimme wurde geheim festgehalten" 7 70
        else
```

```

        echo "`date` | $wahl_zeile | $waehler" >>! .wahlurne
        dialog --clear --msgbox "Vielen Dank.\n
        Ihre Stimme wurde festgehalten" 7 70
    endif
    break
else if ($ergeb != 1) then
    dialog --clear --msgbox "Vielen Dank\n\nProgramm $0 beendet sich" 9
70
    break
endif
end
else
    rm -f ~/auswahl_tmp.$$
    exit (1)
endif
rm -f ~/auswahl_tmp.$$
setterm -default
setterm -clear

```

10.10.2 Das Skript wahl_info

```

#!/bin/tcsh -f
#
#   wahl_info - Skript, welches Informationen zum Stand von
#               laufenden bzw. abgeschlossenen Abstimmungen
#               liefert.
#
set TITEL = "Wahl-Infos"
setterm -store
set dir = $0
set dir = $dir:h
if (" $dir" == "$0") set dir = "."

set themen_datei = $dir/.wahlthemen

if (-e $themen_datei) then
    # awk-Aufruf bewirkt das Ignorieren von Leerzeilen
    # (siehe auch Folgeband dieser Reihe (awk und sed)
    cat $themen_datei | awk '$0 !~ /^[ \t]*$/ ' >~/themen_tmp.$$
    @ themen_zahl = `cat ~/themen_tmp.$$ | wc -l`
else
    @ themen_zahl = 0
endif

if ($themen_zahl == 0) then
    dialog --title "$TITEL" --msgbox "Es steht zur Zeit kein Thema zur Wahl" 10
45
    setterm -default
    setterm -clear
    rm -f ~/themen_tmp.$$
    exit 1

```

```
else
    set menue = `cut -d: -f1 ~/themen_tmp.$$ | awk '{printf("\%s\\n", $0)}' |
n1`
    echo -n '( dialog --title "$TITEL" --clear ' >~/menue.$$
    echo -n '--menu "Folgende Themen stehen zur Wahl" 12 60 ' >>~/menue.$$
    echo -n "$themen_zahl $menue:q " >>~/menue.$$
    echo    ">/dev/tty ) >& ~/tmp.$$" >>~/menue.$$
    source ~/menue.$$
    set ergebn = $status
    setterm -default -clear
    set thema_wahl = `cat ~/tmp.$$`
    setterm -default
    setterm -clear
    rm -f ~/tmp.$$
    rm -f ~/menue.$$
endif

if ( $ergebn == 0 ) then
    set thema = `cut -d: -f2 ~/themen_tmp.$$`
    set thema_datei = $thema[$thema_wahl]
    set thema_zeile = "`awk 'NR == $thema_wahl' ~/themen_tmp.$$ | cut -d: -f1`"
else
    rm -f ~/themen_tmp.$$
    exit (1)
endif
rm -f ~/themen_tmp.$$

if (! -e $dir/$thema_datei ) then
    dialog --title $TITEL --msgbox "Wahldatei --- $dir/$thema_datei --- nicht
vorhanden" 8 60
    setterm -default
    setterm -clear
    exit (1)
else
    cd $dir/$thema_datei
    if (-e .geschlossen) then
        dialog --title $TITEL --msgbox "Wahl fuer --- $thema_zeile ---\\n\\
        bereits abgeschlossen" 8 60
    else
        dialog --title $TITEL --msgbox "Wahl fuer --- $thema_zeile ---\\n\\
        ist noch im Gange" 8 60
    endif
    setterm -default
    setterm -clear
endif

if (-e .wahlurne) then
    echo "Wahlstand am `date`" > ~/tmp.$$
    cut -d'|' -f2 .wahlurne | sort | uniq -c | sort -rn >> ~/tmp.$$
    dialog --clear --title $TITEL --textbox ~/tmp.$$ 20 60
    setterm -default
```

```
        setterm -clear
        rm -f ~/tmp.$$
    else
        dialog --clear --title $TITEL --msgbox "Keine Wahlurne vorhanden" 8 60
        setterm -default
        setterm -clear
        exit (1)
    endif

    if (! -e .geheim) then
        echo "Waehler stimmten wie folgt:\n" >~/tmp.$$
        cut -d"|" -f2- .wahlurne | sort | uniq -c >>~/tmp.$$
        dialog --clear --title $TITEL --textbox ~/tmp.$$ 20 70
        rm -f ~/tmp.$$
        setterm -default
        setterm -clear
    endif
```

Stichwortverzeichnis

!

(bash) 600
(ksh) 253, 298
12, 65, 79, 82, 86, 288, 294, 295, 296, 298,
479, 480, 506, 507, 599, 600, 672

! (csh) 481
!() (ksh) 253, 289
(csh) 480, 504
(ksh) 290
(sh) 67, 68

".." (csh) 504
".." (ksh) 290, 291
".." (sh) 67, 71
"..." (csh) 480

12
(bash) 591
(csh) 480, 484, 564
(ksh) 257, 414
(sh) 21, 210
(tcsh) 680
#! (ksh) 257

\$ 12
\$ (csh) 480, 486
\$#variable (csh) 498
\$(()) (bash) 598, 616
\$() (bash) 598
\$() (ksh) 253, 259
\$(...) 253
\$< (csh) 499
\$?variable (csh) 488, 499
\${#variable} (csh) 498
\${..} 12
\${..} (csh) 480
\${?variable} (csh) 499
\${} (bash) 597
\${} (csh) 497
\${} (ksh) 253, 282
\${10},\${11},... 253, 481
\${variable} (csh) 497
& 12

& (csh) 480, 483
& (ksh) 255
& (sh) 18, 19
&& 12
&& (csh) 480, 483
&& (ksh) 255, 321
&& (sh) 18, 19, 122
&> (bash) 600
'..' 12
'..' (csh) 480
() (ksh) 253
(..) 253
(..) (ksh) 315
() (csh) 480, 481, 517
() (ksh) 318
() (sh) 102
(..) 12
(..) (csh) 480, 517
(bash) 598, 599, 616, 627
(csh) 480, 483, 502
(ksh) 255, 288, 306, 308, 325
(sh) 11, 12, 18, 64, 65, 92
(tcsh) 668
* (bash) 599
* (csh) 480, 502
* (ksh) 288
* (sh) 64
* (tcsh) 668
*() (ksh) 253, 288
+() (ksh) 253, 289
.(bash) 617
.(ksh) 414
.(sh) 157, 210
92, 306, 627
.bash_login (bash) 633
.bash_logout (bash) 633
.bash_profile (bash) 633
.cshrc (csh) 478, 571
.cshrc (tcsh) 640
.login (csh) 478, 572
.login (tcsh) 641

.logout (csh) 479, 573
 .profile
 bash 633
 ksh 437
 sh 223
 .profile (ksh) 252
 .tcshrc (tcsh) 640
 /etc/cshrc (csh) 571
 ; 480, 483
 < (csh) 479, 506
 << 11
 << (csh) 479, 507
 <> (ksh) 253
 = 12
 > (bash) 600
 > (csh) 479, 506
 > (ksh) 294
 > (sh) 80
 > (tcsh) 672
 >! (csh) 481, 506
 >! (tcsh) 672
 >& (bash) 600
 >&- (bash) 600
 >& (csh) 481, 506
 >& (ksh) 296
 >&- (ksh) 298
 >& (sh) 86
 >&- (sh) 86
 >& (tcsh) 672
 >&! (csh) 481, 506
 >&! (tcsh) 672
 >> (bash) 600
 >> (csh) 479, 506, 507
 >> (ksh) 295
 >> (sh) 82
 >> (tcsh) 672
 >>! (csh) 481, 506, 507
 >>! (tcsh) 672
 >>& (csh) 481, 507
 >>& (tcsh) 672
 >>&! (csh) 481, 507
 >>&! (tcsh) 672
 >| (bash) 600
 >| (ksh) 253, 294
 ? (bash) 599
 ? (csh) 480, 502
 ? (ksh) 288
 ? (sh) 64
 ? (tcsh) 668
 ?() (ksh) 253, 288
 @ (csh) 488, 513, 566

@ (tcsh) 680
 @() (ksh) 253, 289
 [!...] 12
 308, 325
 ^ (csh) 481
 `..` 12
 `..` (csh) 480, 485
 {...} (csh) 481
 {...} 12
 {} (csh) 502
 {} (ksh) 318
 {} (sh) 105
 {} (tcsh) 668
 | (csh) 480, 483
 | (sh) 16
 | & (csh) 481
 | & (ksh) 253, 255
 || 12
 || (csh) 480, 483
 || (ksh) 255, 321
 || (sh) 18, 19, 122
 } (sh) 12
 ~ (bash) 597
 ~ (csh) 481, 502
 ~ (ksh) 253, 353
 ~ (tcsh) 668
 “ (bash) 598
 “ (csh) 480, 485
 “ (ksh) 258
 “ (sh) 27
 “ (tcsh) 667
 '...' (csh) 480
 '..' (csh) 504
 '..' (ksh) 290, 291
 '..' (sh) 67, 69

A

Abarbeitung von Kommandozeilen

csh 567
 ksh 416
 sh 211

Ablaufsteuerung (sh) 107

achtdame (csh-Skript) 549

addsuffix (tcsh) 708

AFSUSER (tcsh) 708, 725

Alias

autoload (ksh) 349, 416
 beepcmd (tcsh) 678, 706
 cwdcmd (tcsh) 678, 706

- false (ksh) 349, 416
- functions (ksh) 340, 349, 416
- Funktionen (ksh) 349
- hash (ksh) 349, 416
- history (ksh) 349, 416
- integer (ksh) 349, 416
- Löschen (csh) 539
- Mechanismus 250
- Mechanismus (csh) 535
- nohup (ksh) 349, 416
- periodic (tcsh) 678, 707
- precmd (tcsh) 678, 707
- r (ksh) 349, 416
- shell (tcsh) 707
- stop (ksh) 416
- Substitution (csh) 537
- suspend (ksh) 416
- true (ksh) 349, 416
- type (ksh) 349, 416
- alias (bash) 602, 617
- alias (csh) 536, 564
- alias (ksh) 343, 414
- alias (tcsh) 681
- Alias-Mechanismus (bash) 602
- Alias-Mechanismus (csh) 535
- Alias-Mechanismus (ksh) 343
- Alias-Mechanismus (tcsh) 664
- Alias-Substitution (ksh) 346
- Allgemeines zur Bourne-Shell 11
- Allgemeines zur UNIX-Shell 5
- alloc (tcsh) 681
- allow_null_glob_expansion (bash) 594
- ampm (tcsh) 708
- Anweisung
 - foreach- (csh) 525
 - goto- (csh) 528
 - if- (csh) 517
 - repeat- (csh) 525
 - switch- (csh) 520
 - Umlenkungs- (csh) 506
 - while- (csh) 523
- Anwendungsbeispiele
 - bash 635
 - csh 575
 - dialog 790
 - ksh 447
 - sh 234
 - tcsh 726
- Anzahl von Positionsparametern (csh) 495
- Argumente
 - als Kommandos ausführen (csh) 564
 - argv (csh) 494
 - argv (tcsh) 708
 - Arithmetik (bash) 615
 - Arithmetische Ausdrücke
 - csh 509
 - Arithmetische Auswertungen (ksh) 310
 - Array
 - Element (csh) 498
 - Elementanzahl (csh) 498
 - Index(csh) 498
 - Array (csh) 489
 - Arrays (csh) 489, 498
 - Arrays (ksh) 265, 281, 284
 - at 572
 - Attribute von Variablen (ksh) 276
 - Aufgaben einer Shell 5
 - Aufrufsyntax
 - bash 588
 - csh 568
 - ksh 428
 - sh 216
 - tcsh 639
 - Aufrufsyntax der csh 568
 - Ausdruck
 - arithmetisch (csh) 509
 - Ausdrücke (csh) 509
 - Ausdrücke (ksh) 306, 310
 - Ausdrücke (sh) 92
 - Ausdrücke (tcsh) 673
 - Ausführung
 - bedingte 12
 - bedingte (csh) 480, 517, 520
 - Ausgabe
 - Umlenkung (csh) 506
 - Ausgabeumlenkung (bash) 600
 - Ausgabeumlenkung (csh) 506
 - Ausgabeumlenkung (ksh) 293
 - Ausgabeumlenkung (sh) 78
 - Ausgabeumlenkung (tcsh) 672
 - Auswerten
 - arithmetisch (csh) 509
 - Auswertung von arithmetischen Ausdrücken
 - (ksh) 310
 - Auswertung von Ausdrücken (csh) 509
 - Auswertung von Ausdrücken (ksh) 306
 - Auswertung von Ausdrücken (sh) 92
 - autocorrect (tcsh) 646, 708
 - autoexpand (tcsh) 646, 708
 - autolist (tcsh) 708
 - autoload (ksh) 349, 416
 - Autoload-Funktionen (ksh) 341, 441

autologout (tcsh) 709
 automatische Parameter (csh) 494
 automatische Shell-Variable
 \$ (csh) 494
 * (csh) 494
 argv (csh) 495
 child (csh) 495
 csh 494
 cwd (csh) 495
 status (csh) 495

B

backslash_quote (tcsh) 709
 Bash
 History-Mechanismus 613
 Quoting 591
 bash 585
 .bash_login 633
 .bash_logout 633
 .bash_profile 633
 .profile 633
 Ablaufsteuerung 589
 Alias-Mechanismus 602
 Anwendungsbeispiele 635
 Arithmetik 615
 Aufrufsyntax 588
 Builtin-Editoren 604
 Einfache Kommandos 589
 Erweiterungen 585
 Funktionen 590, 601
 Job-Kontrolle 602
 Listen 589
 Neuheiten 585
 Optionen 588
 Pipeline 589
 Programmiersprachkonstrukte 589
 Prozeß-Substitution 599
 Tilde-Expandierung 597
 BASH (bash) 592
 BASH_VERSION (bash) 592
 bc 235, 729
 bc Tischrechner 738
 bedingte Ausführung 12
 bedingte Ausführung (csh) 480, 517, 520
 Beenden der csh 478
 beepcmd (tcsh) 678, 706
 Begriffe der Shell 9
 Bezeichner 9
 bg (bash) 617

bg (csh) 544, 547, 564
 bg (ksh) 396, 414
 bg (tcsh) 681
 bind (bash) 618
 bindkey (tcsh) 648, 654, 681
 Bourne Again Shell 585
 siehe bash 585
 Bourne-Shell 6
 Allgemeines 11
 Metazeichen 11
 break (bash) 618
 break (csh) 523, 525, 564
 break (ksh) 323, 324, 325, 414
 break (sh) 131, 139, 148, 210
 break (tcsh) 683
 breaksw (tcsh) 683
 builtin (bash) 618
 Builtin-Editoren (bash) 604
 Builtin-Editoren (ksh) 359
 Builtin-Editoren (tcsh) 641
 builtin-Kommando 340
 # (Kommentar) (csh) 564
 @ (csh) 488, 513, 566
 alias (csh) 536, 564
 bg (csh) 547, 564
 break (csh) 523, 525, 564
 cd (csh) 564
 chdir (csh) 564
 continue (csh) 523, 525, 564
 dirs (csh) 553, 564
 echo (csh) 555, 564
 eval (csh) 564
 exec (csh) 565
 exit (csh) 565
 fg (csh) 547, 565
 glob (csh) 555, 565
 hashstat (csh) 557, 565
 history (csh) 565
 jobs (csh) 545, 565
 kill (csh) 546, 565
 limit (csh) 563, 565
 login (csh) 562, 565
 logout (csh) 562, 565
 newgrp (csh) 565
 nice (csh) 561, 565
 nohup (csh) 565
 notify (csh) 548, 565
 onintr (csh) 540, 565
 popd (csh) 553, 565
 print (ksh) 253
 pushd (csh) 553, 565

- read (ksh) 253
- rehash (csh) 557, 565
- set (csh) 487, 565
- setenv (csh) 558, 566
- shift (csh) 487, 559, 566
- source (csh) 566, 573
- stop (csh) 548, 566
- suspend (csh) 549, 566
- time (csh) 566
- umask (csh) 566
- unalias (csh) 539, 566
- unhash (csh) 557, 566
- unlimit (csh) 564, 566
- unset (csh) 566
- unsetenv (csh) 558, 566
- wait (csh) 547, 566
- Builtin-Kommandos
 - bash 617
 - csh 552
 - ksh 400
 - sh 188, 209
 - tcsh 679
- builtins (tcsh) 683
- bye (tcsh) 683

- C**

- case (bash) 590
- case (ksh) 322
- case (sh) 123
- case (tcsh) 683
- cd
 - Such-Directories (csh) 491
- cd (bash) 618
- cd (csh) 564
- cd (ksh) 400, 414
- cd (sh) 210
- cd (tcsh) 683
- cdable_vars (bash) 594
- CDPATH (bash) 594
- CDPATH (csh) 491
- CDPATH (ksh) 267
- CDPATH (sh) 42
- cdpath (tcsh) 709
- cdspell (csh) 574
- chdir (csh) 564
- chdir (tcsh) 683
- child (csh) 495
- chsh 585
- COLUMNS (ksh) 267, 359
- COLUMNS (tcsh) 725
- command (bash) 619
- command (tcsh) 709
- command_oriented_history (bash) 594
- complete (tcsh) 646, 647, 683, 709
- continue (bash) 619
- continue (csh) 523, 525, 564
- continue (ksh) 323, 324, 325, 414
- continue (sh) 149, 210
- continue (tcsh) 688
- copy (csh-Skript) 519
- correct (tcsh) 646, 710
- cron 572
- csh 568
 - .cshrc 571
 - .login 572
 - .logout 573
 - Ablaufsteuerung 517
 - Alias-Mechanismus 535
 - als Login-Shell 478
 - Anwendungsbeispiele 575
 - Array 489
 - Aufrufsyntax 568
 - Beenden 478
 - Debuggen 539
 - Einfache Kommandos 482
 - Fehlersuche 539
 - History-Mechanismus 528
 - Job-Kontrolle 542
 - Kommandosubstitution 485
 - Kommentar 484
 - Listen 482, 483
 - Metazeichen 479
 - Neuheiten 477
 - Optionen 568
 - Parameter 486
 - Pipeline 482, 483
 - Positionsparameter 481
 - Programmiersprachkonstrukte 517
 - Quoting 504
 - Signalbehandlung 540
 - Skript 485
 - Starten 478
- C-Shell 6, 477
 - Neuheiten 477
 - siehe csh
- cwd (csh) 495
- cwd (tcsh) 710
- cwdcmd (tcsh) 678, 706

D

Datei

- .cshrc (csh) 571
- .logout (csh) 573
- .profile (ksh) 252
- /etc/cshrc (csh) 571
- /etc/passwd 7
- Environment- (ksh) 252
- mailbox- (csh) 492
- Paßwort- 7
- Scratch- (csh) 574
- zum Lesen und Schreiben eröffnen
(ksh) 253

Dateideskriptor (csh) 506

Dateinamenexpandierung 11

- !(...) 253
- *(csh) 502
- *(...) 253
- +(...) 253
- ? (csh) 502
- ?(...) 253
- @(...) 253
- [...] (csh) 502
- {...} (csh) 481, 502
- ~ (csh) 502
- ksh 253

Dateinamenexpandierung (bash) 596, 599

Dateinamenexpandierung (csh) 480, 501

Dateinamenexpandierung (ksh) 287

Dateinamenexpandierung (sh) 64

Dateinamenexpandierung (tcsh) 668

dc 729

- Aufrufsyntax 730

- Eingaben 730

dc Tischrechner 729

Debuggen

- C-Shell 539

Debuggen (csh) 539

Debuggen (ksh) 385

Debuggen (sh) 168

declare (bash) 619

default (tcsh) 688

delfang (csh-Skript) 541

dextract (tcsh) 710

dialog

- Anwendungsbeispiele 790

dialog (Linux) 761

dialog Kommando, 761

Dialogboxen (Linux) 761

Dialogboxen 761

Directory

- Home- (csh) 492

- Wechseln (csh) 564

Directory-Stack (csh) 553

dirs (bash) 620

dirs (csh) 553, 564

dirs (ksh) 335

dirs (tcsh) 669, 689

dirsfile (tcsh) 669, 710

dirstack (tcsh) 669, 710

DISPLAY (tcsh) 725

dunique (tcsh) 710

E

echo (bash) 620

echo (csh) 491, 540, 555, 564

echo (ksh) 414

echo (sh) 210

echo (tcsh) 689, 710

echo_style (tcsh) 711

echotc (tcsh) 690

edit (tcsh) 711

EDITOR (ksh) 267

EDITOR (tcsh) 725

Editoren

- Builtin- (bash) 604

- Builtin- (ksh) 359

- Builtin- (tcsh) 641

Eigenschaften der Shell 7

Einfache Kommandos 13

Einfache Kommandos (bash) 589

Einfache Kommandos (csh) 482

Einfache Kommandos (ksh) 255

Einfache Kommandos (sh) 13

Einfache Kommandos (tcsh) 670

Eingabe

- Umlenkung (csh) 506

Eingaben

- dc- 730

Eingabeumlenkung (bash) 600

Eingabeumlenkung (csh) 506

Eingabeumlenkung (ksh) 293

Eingabeumlenkung (sh) 78

Eingabeumlenkung (tcsh) 672

Element

- eines Arrays (csh) 498

ellipsis (tcsh) 711

emacs (ksh) 361

enable (bash) 620

ENV (bash) 594
 env (csh) 558
 ENV (ksh) 252, 267, 439
 env (sh) 61
 Environment-Datei (ksh) 252, 439
 ERRNO (ksh) 273
 EUID (bash) 592
 eval (bash) 620
 eval (csh) 564
 eval (ksh) 414
 eval (sh) 190, 210
 eval (tcsh) 691
 exec (bash) 620
 exec (csh) 565
 exec (ksh) 414
 exec (sh) 195, 210
 exec (tcsh) 691
 exit (bash) 620
 exit (csh) 565
 exit (ksh) 414
 exit (sh) 210
 exit (tcsh) 691
 Exit-Status
 Vordergrund-Kommando (csh) 495
 exit-Status
 case (ksh) 323
 case (sh) 128
 for (sh) 148
 Funktion (sh) 165
 if (ksh) 322
 if (sh) 122
 Kommando 14
 Kommando (csh) 482
 Pipeline (csh) 483
 Pipeline (sh) 18
 until (ksh) 324
 until (sh) 140
 while (ksh) 323
 while (sh) 133
 Expandierung
 Dateinamen 11
 Dateinamen (bash) 596, 599
 Dateinamen (csh) 480, 501
 Dateinamen (ksh) 287
 Dateinamen (sh) 64
 Dateinamen (tcsh) 668
 Metazeichen 11
 Metazeichen (csh) 480
 Tilde- (bash) 597
 Tilde- (ksh) 250, 353
 export (bash) 621

export (ksh) 414
 export (sh) 59, 61, 210
 expr (sh) 96

F

false (ksh) 349, 416
 fc (bash) 621
 fc (ksh) 354, 414
 FCEDIT (bash) 594
 FCEDIT (ksh) 268
 Fehlersuche
 C-Shell 539
 Fehlersuche (csh) 539
 Fehlersuche (ksh) 385
 Fehlersuche (sh) 168
 fg (bash) 621
 fg (csh) 544, 547, 565
 fg (ksh) 397, 414
 fg (tcsh) 691
 IGNORE (bash) 594
 ignore (tcsh) 645, 711
 filec (tcsh) 711
 filetest (tcsh) 691
 for (bash) 590
 for (ksh) 325
 for (sh) 140
 foreach (csh) 525
 foreach (tcsh) 691
 foreach-Anweisung (csh) 525
 FPATH (ksh) 268, 342
 frei wählbare Shell-Variable (csh) 488
 functions (ksh) 340, 349, 416
 Funktion (bash) 590, 601
 Funktion (sh) 154
 Funktion
 Aliase (ksh) 349
 Autoload- (ksh) 341, 441
 ksh 332

G

geklammerte Kommandos 13
 geklammerte Kommandos (csh) 517
 Geschweifte Klammern 12
 Geschweifte Klammern (csh) 480
 Geschweifte Klammern (ksh) 318
 Geschweifte Klammern (sh) 105
 getopts (bash) 621

getopts (ksh) 414
 getopts (sh) 196, 210
 getspath (tcsh) 691
 getxvers (tcsh) 691
 gid (tcsh) 711
 glob (csh) 555, 565
 glob (tcsh) 692
 glob_dot_filenames (bash) 594
 gmacs (ksh) 361
 goto (csh) 528
 goto (tcsh) 692
 goto-Anweisung (csh) 528
 group (tcsh) 711, 725
 Gültigkeitsbereich
 Shell-Variable (sh) 56

H

hash (bash) 621
 hash (ksh) 349, 416
 hash (sh) 204, 210
 Hashing-Mechanismus (csh) 556
 hashstat (csh) 557, 565
 hashstat (tcsh) 692
 help (bash) 621
 hexziff (csh-Skript) 521
 hexziff2 (csh-Skript) 522
 Hier-Dokument 11
 Quoting (csh) 507
 Hier-Dokument (csh) 479, 507
 Hier-Dokument (ksh) 295
 Hier-Dokument (sh) 82
 Hintergrundjob 12
 Hintergrundjob (csh) 480
 Hintergrundjobs
 in Vordergrund bringen (csh) 565
 Hintergrund-Kommando
 PID (csh) 495
 Prozeßnummer (csh) 495
 histchars (bash) 594
 histchars (csh) 491
 histchars (tcsh) 711
 HISTCMD (bash) 592
 HISTCONTROL (bash) 594
 histdup (tcsh) 712
 HISTFILE (bash) 594
 HISTFILE (ksh) 268
 histfile (tcsh) 712
 HISTFILESIZE (bash) 595

histlit (tcsh) 712
 History
 -Mechanismus (ksh) 250
 -Puffer (csh) 492, 529
 history (bash) 622
 history (csh) 492, 529, 565
 history (ksh) 349, 416
 history (tcsh) 692, 712
 history_control (bash) 594
 History-Mechanismus (bash) 613
 History-Mechanismus (csh) 528
 History-Mechanismus (ksh) 353
 History-Mechanismus (tcsh) 659
 History-Puffer 528
 History-Substitution 530
 HISTSIZE (bash) 595
 HISTSIZE (ksh) 268
 HOME (bash) 595
 HOME (csh) 492
 HOME (ksh) 268
 HOME (sh) 41
 HOME (tcsh) 725
 HOME (tcsh) 712
 Home-Directory (csh) 492
 HOST (tcsh) 725
 HOSTFILE (bash) 595
 hostname_completion_file (bash) 595
 HOSTTYPE (bash) 592
 HOSTTYPE (tcsh) 725
 HPATH (tcsh) 725
 hup (tcsh) 693

I

if (bash) 589
 if (csh) 517
 if (ksh) 321
 if (sh) 112
 if (tcsh) 693
 if-Anweisung (csh) 517
 IFS (bash) 595
 IFS (ksh) 268
 IFS (sh) 44
 IGNOREEOF (bash) 595
 ignoreeof (csh) 492
 ignoreeof (tcsh) 713
 Index
 eines Arrays (csh) 498
 Infix-Schreibweise 729

inlib (tcsh) 694
inputmode (tcsh) 713
INPUTRC (bash) 595
integer (ksh) 349, 416

J

Job
 Hintergrund- 12
 Hintergrund- (csh) 480
Job-Kontrolle 250
Job-Kontrolle (bash) 602
Job-Kontrolle (csh) 542
Job-Kontrolle (ksh) 391
Job-Kontrolle (sh) 229
Job-Kontrolle (tcsh) 677
Jobs
 Signale schicken (csh) 565
jobs (bash) 622
jobs (csh) 543, 545, 565
jobs (ksh) 393, 414
jobs (tcsh) 694

K

kill (bash) 623
kill (csh) 546, 565
kill (ksh) 389, 395, 414
kill (tcsh) 694
Klammer
 geschweifte 12
 geschweifte (csh) 480
 geschweifte (ksh) 318
 geschweifte (sh) 105
 runde 12
 runde (csh) 480, 517
 runde (ksh) 318
 runde (sh) 102
Klammerung (csh) 517
Klammerung (ksh) 318
Klammerung (sh) 102
Kommando
 (bash) 617
 (ksh) 414
 (sh) 189, 210
 (tcsh) 680
(csh) 564
(Kommentar) (csh) 564
(ksh) 414

(sh) 210
(tcsh) 680
((..)) (ksh) 315
.(bash) 617
.(ksh) 414
.(sh) 157, 210
@ (csh) 488, 513, 566
@ (tcsh) 680
Ablaufsteuerung (bash) 589
Ablaufsteuerung (csh) 517
Ablaufsteuerung (ksh) 319
Ablaufsteuerung (sh) 107
alias (bash) 602, 617
alias (csh) 536, 564
alias (ksh) 343, 414
alias (tcsh) 681
alloc (tcsh) 681
Ausführen der Argumente als (csh) 564
bc 235, 729, 738
bg (bash) 617
bg (csh) 544, 547, 564
bg (ksh) 396, 414
bg (tcsh) 681
bind (bash) 618
bindkey (tcsh) 648, 654, 681
break (bash) 618
break (csh) 523, 525, 564
break (ksh) 323, 324, 325, 414
break (sh) 131, 139, 148, 210
break (tcsh) 683
breaksw (tcsh) 683
Builtin (bash) 618
Builtin- (bash) 617
Builtin- (csh) 552
Builtin- (ksh) 400
Builtin- (sh) 188, 209
Builtin- (tcsh) 679, 683
bye (tcsh) 683
case (bash) 590
case (ksh) 322
case (sh) 123
case (tcsh) 683
cd (bash) 618
cd (csh) 564
cd (ksh) 400, 414
cd (sh) 210
cd (tcsh) 683
chdir (csh) 564
chdir (tcsh) 683
command (bash) 619
complete (tcsh) 647, 683

- continue (bash) 619
- continue (csh) 523, 525, 564
- continue (ksh) 323, 324, 325, 414
- continue (sh) 149, 210
- continue (tcsh) 688
- csh 568
- dc 729
- declare (bash) 619
- default (tcsh) 688
- dialog (Linux) 761
- dirs (bash) 620
- dirs (csh) 553, 564
- dirs (tcsh) 669, 689
- echo (bash) 620
- echo (csh) 555, 564
- echo (ksh) 414
- echo (sh) 210
- echo (tcsh) 689
- echotc (tcsh) 690
- Einfach 13
- einfaches 13
- einfaches (csh) 482
- enable (bash) 620
- env (csh) 558
- env (sh) 61
- eval (bash) 620
- eval (csh) 564
- eval (ksh) 414
- eval (sh) 190, 210
- eval (tcsh) 691
- exec (bash) 620
- exec (csh) 565
- exec (ksh) 414
- exec (sh) 195, 210
- exec (tcsh) 691
- exit (bash) 620
- exit (csh) 565
- exit (ksh) 414
- exit (sh) 210
- exit (tcsh) 691
- Exit-Status (csh) 495
- exit-Status (csh) 482
- export (bash) 621
- export (ksh) 414
- export (sh) 59, 61, 210
- expr (sh) 96
- fc (bash) 621
- fc (ksh) 354, 414
- fg (bash) 621
- fg (csh) 544, 547, 565
- fg (ksh) 397, 414
- fg (tcsh) 691
- filetest (tcsh) 691
- for (bash) 590
- for (ksh) 325
- for (sh) 140
- foreach (csh) 525
- foreach (tcsh) 691
- geklammertes 13
- geklammertes (csh) 517
- getopts (bash) 621
- getopts (ksh) 414
- getopts (sh) 196, 210
- getspath (tcsh) 691
- glob (csh) 555, 565
- glob (tcsh) 692
- goto (csh) 528
- goto (tcsh) 692
- hash (bash) 621
- hash (sh) 204, 210
- hashstat (csh) 557, 565
- hashstat (tcsh) 692
- help (bash) 621
- history (bash) 622
- history (csh) 529, 565
- history (tcsh) 692
- hup (tcsh) 693
- if (bash) 589
- if (csh) 517
- if (ksh) 321
- if (sh) 112
- if (tcsh) 693
- inlib (tcsh) 694
- jobs (bash) 622
- jobs (csh) 543, 545, 565
- jobs (ksh) 393, 414
- jobs (tcsh) 694
- kill (bash) 623
- kill (csh) 546, 565
- kill (ksh) 389, 395, 414
- kill (tcsh) 694
- Klammerung (csh) 517
- Klammerung (ksh) 318
- Klammerung (sh) 102
- let (bash) 623
- let (ksh) 315, 414
- limit (csh) 563, 565
- limit (tcsh) 694
- Liste (ksh) 255
- local (bash) 623
- log (tcsh) 695
- login (csh) 562, 565

- login (tcsh) 695
- logout (bash) 623
- logout (csh) 562, 565
- logout (tcsh) 695
- ls-F (tcsh) 695
- migrate (tcsh) 696
- newgrp (csh) 565
- newgrp (ksh) 414
- newgrp (sh) 210
- newgrp (tcsh) 696
- nice (csh) 561, 565
- nice (tcsh) 696
- nohup (csh) 565
- nohup (tcsh) 697
- notify (csh) 548, 565
- notify (tcsh) 678, 697
- Null- (bash) 617
- Null- (ksh) 414
- Null- (sh) 189
- onintr (csh) 540, 565
- onintr (tcsh) 679, 697
- popd (bash) 623
- popd (csh) 553, 565
- popd (tcsh) 669, 697
- print (ksh) 304, 414
- printenv (csh) 558
- printenv (tcsh) 697
- Punkt- (bash) 617
- Punkt- (ksh) 414
- Punkt- (sh) 157
- pushd (bash) 623
- pushd (csh) 553, 565
- pushd (tcsh) 669, 698
- pwd (bash) 624
- pwd (ksh) 415
- pwd (sh) 210
- read (bash) 624
- read (ksh) 265, 301, 415
- read (sh) 39, 210
- readonly (bash) 624
- readonly (ksh) 415
- readonly (sh) 208, 210
- rehash (csh) 557, 565
- rehash (tcsh) 698
- repeat (csh) 525
- repeat (tcsh) 699
- return (bash) 601, 624
- return (ksh) 340, 415
- return (sh) 165, 210
- rootnode (tcsh) 699
- rsh 227
- Rückgabewert (csh) 482
- sched (tcsh) 699
- select (bash) 590
- select (ksh) 326
- set (bash) 624
- set (csh) 487, 565
- set (ksh) 263, 403, 415
- set (sh) 32, 45, 169, 206, 210
- set (tcsh) 700
- setenv (csh) 487, 558, 566
- setenv (tcsh) 701
- setpath (tcsh) 701
- setspath (tcsh) 701
- settc (tcsh) 701
- setty (tcsh) 701
- setxvers (tcsh) 702
- shift (bash) 626
- shift (csh) 487, 559, 566
- shift (ksh) 263, 415
- shift (sh) 34, 210
- shift (tcsh) 703
- shl 229
- source (bash) 626
- source (csh) 560, 566, 573
- source (tcsh) 703
- stop (csh) 548, 566
- stop (tcsh) 703
- Suchdirectories (csh) 493
- suspend (bash) 627
- suspend (csh) 549, 566
- suspend (tcsh) 703
- switch (csh) 520
- switch (tcsh) 703
- telltc (tcsh) 704
- test (bash) 627
- test (ksh) 306, 415
- test (sh) 92, 210
- time (csh) 561, 566
- time (ksh) 331
- time (tcsh) 704
- times (bash) 629
- times (ksh) 411, 415
- times (sh) 209, 211
- trap (bash) 629
- trap (ksh) 389, 415
- trap (sh) 182, 211
- type (bash) 630
- type (sh) 164, 211
- typeset (bash) 631
- typeset (ksh) 276
- typset (ksh) 415

- ulimit (bash) 631
- ulimit (ksh) 412, 415
- ulimit (sh) 209, 211
- umask (bash) 632
- umask (csh) 566
- umask (ksh) 415
- umask (sh) 211
- umask (tcsh) 704
- unalias (bash) 632
- unalias (csh) 539, 566
- unalias (ksh) 346, 415
- unalias (tcsh) 704
- uncomplete (tcsh) 705
- unhash (csh) 557, 566
- unhash (tcsh) 705
- unlimit (csh) 564, 566
- unlimit (tcsh) 705
- unset (bash) 632
- unset (csh) 566
- unset (ksh) 265, 341, 415
- unset (sh) 38, 61, 211
- unset (tcsh) 705
- unsetenv (csh) 558, 566
- unsetenv (tcsh) 705
- until (bash) 590
- until (ksh) 324
- until (sh) 135
- ver (tcsh) 705
- wait (bash) 632
- wait (csh) 547, 566
- wait (ksh) 396, 415
- wait (sh) 211
- wait (tcsh) 705
- watchlog (tcsh) 706
- whence (ksh) 402, 415
- where (tcsh) 706
- which (tcsh) 706
- while (bash) 590
- while (csh) 523
- while (ksh) 323
- while (sh) 129
- while (tcsh) 706
- xargs 220
- Kommandoklammerung (csh) 517
- Kommandoliste 12, 13
- Kommandoliste (csh) 480, 482, 483
- Kommandoliste (ksh) 255
- Kommandosetenv (csh) 558
- Kommandosubstitution 12
 - csh 485
 - ksh 253
 - Metazeichen (sh) 29
 - Quoting (csh) 485
 - Quoting (sh) 75
 - Schachtelung (sh) 30
- Kommandosubstitution (bash) 598
- Kommandosubstitution (csh) 480, 485
- Kommandosubstitution (ksh) 258
- Kommandosubstitution (sh) 27
- Kommandosubstitution (tcsh) 667
- Kommandozeile 9
 - Abarbeitung (csh) 567
 - Abarbeitung (ksh) 416
 - Abarbeitung (sh) 211
 - Abarbeitungsreihenfolge (csh) 567
 - zu lang 219
- Kommentar (bash) 591
- Kommentar (csh) 484
- Kommentar (ksh) 257
- Kommentar (sh) 21
- Kommentarzeichen 12
- Kommentarzeichen (csh) 480
- Kommunikation
 - Kind- und Elternprozeß (ksh) 253
- Kompatibilität 249, 477
- Konstanten (ksh) 310
- Ko-Prozeß (ksh) 253
- Korn-Shell 6, 249
 - Neuheiten 249
 - siehe ksh 249
- ksh
 - .profile 437
 - Ablaufsteuerung 319
 - Alias-Mechanismus 343
 - Alias-Substitution 346
 - als Login-Shell 251
 - Anwendungsbeispiele 447
 - Arithmetische Ausdrücke (ksh) 310
 - Aufrufsyntax 428
 - Autoload-Funktionen 341, 441
 - Builtin-Editoren 359
 - Debuggen 385
 - Einfache Kommandos 255
 - emacs 361
 - Environment-Datei 439
 - Erweiterungen 249
 - Fehlersuche 385
 - Funktionen 332
 - gmacs 361
 - History-Mechanismus 353
 - Job-Kontrolle 391
 - Konstanten 310

- Listen 255
- Metazeichen 253
- neue Kommandosubstitution 253
- Neuheiten 249
- Pipeline 255
- Positionsparameter 253
- Programmiersprachkonstrukte 319
- Quoting 290
- Signalbehandlung 387
- Starten 251
- Tilde-Expandierung 353
- Tracked Aliase 347
- Umgebung 428
- Variablen (arithmetisch) 310
- vi 371
- Vordefinierte Aliase 349, 416

L

- Länge
 - Array (csh) 498
- LANG (tcsh) 725
- Layers (sh) 230
- LC_CTYPE (tcsh) 725
- Leerzeichen 12
- Leerzeichen (csh) 480
- Lesen einer Eingabezeile (csh) 499
- let (bash) 623
- let (ksh) 315, 414
- limit (csh) 563, 565
- limit (tcsh) 694
- LINENO (bash) 593
- LINENO (ksh) 273
- LINES (ksh) 269
- LINES (tcsh) 725
- Linux 2
- Liste
 - Kommando- 12, 13
 - Kommando- (csh) 480, 482, 483
 - Kommando- (ksh) 255
- Liste (sh) 20
- Listen (Bash) 589
- Listen (csh) 482, 483
- Listen (ksh) 255
- Listen (sh) 13, 18
- Listen (tcsh) 670
- listflags (tcsh) 713
- listjobs (tcsh) 677, 713
- listlinks (tcsh) 713

- listmax (tcsh) 647, 714
- listmaxrows (tcsh) 647, 714
- local (bash) 623
- loesch (csh-Skript) 521
- log (tcsh) 695
- login (csh) 562, 565
- login (tcsh) 695
- loginsh (tcsh) 714
- Login-Shell 7
 - csh 478
 - ksh 251
- LOGNAME (sh) 44
- logout (bash) 623
- logout (csh) 562, 565
- logout (tcsh) 695, 714
- ls-F (tcsh) 695

M

- MACHTYPE (tcsh) 725
- MAIL (bash) 595
- MAIL (csh) 492
- MAIL (ksh) 269
- MAIL (sh) 42
- MAIL (tcsh) 714
- mailbox
 - Dateiname (csh) 492
- MAILCHECK (bash) 595
- MAILCHECK (ksh) 269
- MAILCHECK (sh) 43
- MAILPATH (bash) 595
- MAILPATH (ksh) 269
- MAILPATH (sh) 42
- MAILWARNING (bash) 595
- matchbeep (tcsh) 646, 714
- Metazeichen 86
 - 11, 11, 12, 18, 79, 82, 86, 86, 255, 294, 295,
 - 296, 296, 298, 479, 479, 480, 483, 506,
 - 507, 600, 600, 600, 600, 600, 672, 672
 - (csh) 481
 - (bash) 600
 - (ksh) 253, 298
 - ! (csh) 481
 - !() (ksh) 253, 289
 - !(...) 253
 - (csh) 480, 504
 - (ksh) 290
 - (sh) 67, 68
 - ".." (csh) 480, 504

- ".." (ksh) 290, 291
- ".." (sh) 67, 71
- # 12
- #(bash) 591
- #(csh) 480, 484
- #(ksh) 257
- #(sh) 12, 21
- #!(ksh) 257
- \$ 12
- \$(csh) 480, 486
- \$(sh) 12
- \$() (bash) 598, 616
- \$() (bash) 598
- \$() (ksh) 253, 259
- \${..} 12
- \${..} (csh) 480
- \${} (bash) 597
- \${} (csh) 497
- \${} (ksh) 253
- & 12
- &(csh) 480, 483
- &(ksh) 255
- &(sh) 12, 18, 19
- && 12
- &&(csh) 480, 483
- &&(ksh) 255, 321
- &&(sh) 12, 18, 19, 122
- &> (bash) 600
- '..' 12
- '..' (csh) 480
- () (ksh) 253
- () (csh) 480, 481, 517
- () (ksh) 318
- () (sh) 12, 102
- (..) 12
- (..) (csh) 480, 517
- * 11
- *(bash) 599
- *(csh) 480, 502
- *(ksh) 288
- *(sh) 11, 64
- *(tcsh) 668
- *() (ksh) 253, 288
- +() (ksh) 253, 289
- +(...) 253
- < 11
- <(csh) 479, 506
- << 11
- <<(csh) 479, 507
- <> (ksh) 253
- = 12
- > 11
- >(bash) 600
- >(csh) 479, 506
- >(ksh) 294
- >(sh) 11, 80
- >(tcsh) 672
- >!(csh) 481, 506
- >!(tcsh) 672
- >&(bash) 600
- >&-(bash) 600
- >&(csh) 481, 506
- >&(ksh) 296
- >&-(ksh) 298
- >&(sh) 86
- >&-(sh) 86
- >&(tcsh) 672
- >&!(csh) 481, 506
- >&!(tcsh) 672
- >> 11
- >>(bash) 600
- >>(csh) 479, 506, 507
- >>(ksh) 295
- >>(sh) 11, 82
- >>(tcsh) 672
- >>!(csh) 481, 506, 507
- >>!(tcsh) 672
- >>&(csh) 481, 507
- >>&(tcsh) 672
- >>&!(csh) 481, 507
- >>&!(tcsh) 672
- >| (bash) 600
- >| (ksh) 253, 294
- ? 11
- ? (bash) 599
- ? (csh) 480, 502
- ? (ksh) 288
- ? (sh) 11, 64
- ? (tcsh) 668
- ?() (ksh) 253, 288
- ?(...) 253
- @() (ksh) 253, 289
- @(...) 253
- [!...] 12
- [...] 11
- [...] (csh) 480, 502
- ^ (csh) 481
- `` 12
- `` (csh) 480, 485
- { 12

- {...} (csh) 502
- {..} 12
- { } (ksh) 318
- { } (sh) 105
- { } (tcsh) 668
- | 11
- | (csh) 480
- | (sh) 11, 16
- | & (csh) 481
- | & (ksh) 253, 255
- || 12
- || (csh) 480, 483
- || (ksh) 255, 321
- || (sh) 12, 18, 19, 122
- ~ (bash) 597
- ~ (csh) 481, 502
- ~ (ksh) 253, 353
- ~ (tcsh) 668
- ~() (ksh) 253
- “ (bash) 598
- “ (csh) 480, 485
- “ (ksh) 258
- “ (sh) 12, 27
- “ (tcsh) 667
- ‘..’ (csh) 480, 504
- ‘..’ (ksh) 290, 291
- ‘..’ (sh) 67, 69
- Ausschalten 12
- Ausschalten (csh) 480, 504
- Ausschalten (ksh) 290
- Ausschalten (sh) 67
- Bourne-Shell 11
- csh 479
- in C-Shell 479
- in Korn-Shell 253
- Kommandosubstitution (sh) 12, 29
- Kommentar 12
- Kommentar (csh) 480
- Kommentar (sh) 12
- ksh 253
- Quoting (csh) 504
- Quoting (ksh) 290
- Quoting (sh) 67
- Subshell (sh) 12
- Tilde (csh) 481, 502
- Tilde (ksh) 253
- Tilde (tcsh) 668
- migrate (tcsh) 696
- Modifikatoren (csh) 500, 531
- Monitor 6

N

- namkill (csh-Skript) 527
- Neuzeile-Zeichen 12
- Neuzeile-Zeichen (csh) 480
- Neuheiten der C-Shell 477
- Neuheiten der Korn-Shell 249
- newgrp (csh) 565
- newgrp (ksh) 414
- newgrp (sh) 210
- newgrp (tcsh) 696
- nice (csh) 561, 565
- nice (tcsh) 696
- no_exit_on_failed_exec (bash) 595
- nobeeep (tcsh) 646, 715
- noclobber 506
- noclobber (bash) 595
- noclobber (csh) 492
- noclobber (tcsh) 715
- noglob (csh) 492, 502
- noglob (tcsh) 715
- nohup (csh) 565
- nohup (ksh) 349, 416
- nohup (tcsh) 697
- nokanji (tcsh) 715
- nolinks (bash) 595
- nonomatch (csh) 493
- nonomatch (tcsh) 715
- NOREBIND (tcsh) 725
- noestat (tcsh) 715
- notify (bash) 595
- notify (csh) 493, 548, 565
- notify (tcsh) 678, 697, 716
- Null-Kommando (sh) 189

O

- oid (tcsh) 716
- OLDPWD (bash) 593
- OLDPWD (ksh) 274
- onintr (csh) 540, 565
- onintr (tcsh) 679, 697
- Operatoren
- Umlenkungs- (csh) 506
- OPTARG (bash) 593
- OPTARG (ksh) 274
- OPTERR (bash) 596
- OPTIND (bash) 593
- OPTIND (ksh) 274

Optionen

 csh 568
 ksh 430
 Setzen (sh) 206
 sh 206, 216

OSTYPE (bash) 593

OSTYPE (tcsh) 726

owd (tcsh) 716

P

Parameter

 automatische (csh) 494
 bash 591
 csh 486
 ksh 261
 Positions- 12
 Positions- (bash) 591
 Positions- (csh) 480, 486
 Positions- (ksh) 261
 Positions- (sh) 31
 Schlüsselwort- (csh) 487
 Schlüsselwort- (ksh) 264
 Schlüsselwort- (sh) 35
 sh 31
 Verschieben (csh) 487

Parameter (csh) 486

Paramtersubstitution (csh) 497

Paßwort-Datei 7

PATH (bash) 596

PATH (csh) 493

PATH (ksh) 252, 269

PATH (sh) 41

PATH (tcsh) 716, 726

pattern

 Dateinamen-Expandierung (csh) 501

pattern (csh) 501

periodic (tcsh) 678, 707

PID

 des letzten Hintergrund-Jobs (csh) 495

 Shell (csh) 494

Pipe 11

 Zweiwege- (ksh) 253

Pipe (csh) 480

Pipeline 13

 exit-Status (csh) 483

Pipeline (bash) 589

Pipeline (csh) 482, 483

Pipeline (ksh) 255

Pipeline (sh) 13, 16, 20

Pipeline (tcsh) 670

Polnische Notation 729

popd (bash) 623

popd (csh) 553, 565

popd (ksh) 335

popd (sh) 166

popd (tcsh) 669, 697

Positionsparameter 12, 480

 Anzahl (csh) 495

 csh 481, 486

 ksh 253

 Verschieben (csh) 487

 Zugriff auf alle (csh) 494

Positionsparameter (bash) 591

Positionsparameter (csh) 486

Positionsparameter (ksh) 261

Positionsparameter (sh) 31

Postfix-Schreibweise 729

PPID (bash) 593

PPID (ksh) 274

precmd (tcsh) 678, 707

Primär-Promptstring (csh) 493, 529

print (ksh) 253, 304, 414

printenv (csh) 558

printenv (tcsh) 697

printexitvalue (tcsh) 716

Programme

 Suchdirectories (csh) 493

Programmiersprachkonstrukte (bash) 589

Programmiersprachkonstrukte (ksh) 319

Programmiersprachkonstrukte (sh) 107

Prompt

 Primär-(csh) 493, 529

prompt (csh) 493, 529

prompt (tcsh) 716

PROMPT_COMMAND (bash) 593, 603

prompt2 (tcsh) 718

prompt3 (tcsh) 719

promptchars (tcsh) 719

Prozeßnummer

 des letzten Hintergrund-Jobs (csh) 495

 Shell (csh) 494

Prozeß-Substitution (bash) 599

PS1 (bash) 596, 603

PS1 (ksh) 270

PS1 (sh) 43

PS2 (bash) 596

PS2 (ksh) 270

PS2 (sh) 43

PS3 (bash) 596

PS3 (ksh) 270, 326

PS4 (bash) 596
 PS4 (ksh) 270
 Puffer
 History- (csh) 492, 528, 529
 pushd (bash) 623
 pushd (csh) 553, 565
 pushd (ksh) 335
 pushd (sh) 166
 pushd (tcsh) 669, 698
 pushdsilent (tcsh) 719
 pushdtohome (tcsh) 719
 PWD (bash) 593, 624
 PWD (ksh) 274, 415
 PWD (sh) 210
 PWD (tcsh) 726

Q

Quoting
 Hier-Dokument (csh) 507
 Kommandosubstitution (csh) 485
 Kommandosubstitution (sh) 75
 Quoting (bash) 591
 Quoting (ccsh) 504
 Quoting (csh) 504
 Quoting (ksh) 290
 Quoting (sh) 67

R

r (ksh) 349, 416
 RANDOM (bash) 593
 RANDOM (ksh) 274
 read (bash) 624
 read (ksh) 253, 265, 301, 415
 read (sh) 39, 210
 readonly (bash) 624
 readonly (ksh) 415
 readonly (sh) 208, 210
 recexact (tcsh) 646, 719
 rechne (ksh-Skript) 254
 recognize_only_executables (tcsh) 647, 719
 rehash (csh) 557, 565
 rehash (tcsh) 698
 Reihenfolge
 Abarbeitung der Kommandozeile
 (csh) 567
 REMOTEHOST (tcsh) 726
 repeat (csh) 525

repeat (tcsh) 699
 repeat-Anweisung (csh) 525
 REPLY (bash) 593
 REPLY (ksh) 274, 327
 Restricted Shell (rhs) 227
 return (bash) 601, 624
 return (ksh) 340, 415
 return (sh) 165, 210
 rmstar (tcsh) 719
 rootnode (tcsh) 699
 rsh 227
 Rückgabewert
 eines Kommandos (csh) 482
 Runde Klammern 12
 Runde Klammern (csh) 480, 517
 Runde Klammern (ksh) 318
 Runde Klammern (sh) 102

S

savedirs (tcsh) 669, 719
 savehist (csh) 493
 savehist (tcsh) 719
 sched (tcsh) 699, 720
 Schleife
 Ausführung wiederholen (csh) 564
 foreach- (csh) 525
 goto- (csh) 528
 repeat- (csh) 525
 Verlassen (csh) 564
 while- (csh) 523
 Schlüsselwort
 -Parameter (csh) 487
 Schlüsselwort-Parameter (csh) 487
 Schlüsselwort-Parameter (ksh) 264
 Schlüsselwort-Parameter (sh) 35
 Scratch-Datei (csh) 574
 SECONDS (bash) 593
 SECONDS (ksh) 274
 select (bash) 590
 select (ksh) 326
 set (bash) 624
 set (csh) 487, 565
 set (ksh) 263, 403, 415
 set (sh) 32, 45, 169, 206, 210
 set (tcsh) 700
 setenv (csh) 487, 558, 566
 setenv (tcsh) 701
 setpath (tcsh) 701
 setspath (tcsh) 701

- settc (tcsh) 701
- setty (tcsh) 701
- setxvers (tcsh) 702
- SHACCT (ksh) 270
- SHACCT (sh) 44
- Shell
 - .bash_login (bash) 633
 - .bash_logout (bash) 633
 - .bash_profile (bash) 633
 - .cshrc (csh) 571
 - .cshrc (tcsh) 640
 - .login (csh) 572
 - .login (tcsh) 641
 - .logout (csh) 573
 - .profile (bash) 633
 - .profile (ksh) 437
 - .profile (sh) 223
 - .tcshrc (tcsh) 640
 - Ablaufsteuerung (bash) 589
 - Ablaufsteuerung (csh) 517
 - Ablaufsteuerung (ksh) 319
 - Ablaufsteuerung (sh) 107
 - Allgemeines 5
 - Anwendungsbeispiele (bash) 635
 - Anwendungsbeispiele (csh) 575
 - Anwendungsbeispiele (ksh) 447
 - Anwendungsbeispiele (Sh) 234
 - Anwendungsbeispiele (tcsh) 726
 - Aufgaben 5
 - Aufgaben einer 5
 - Aufrufsyntax (csh) 568
 - Aufrufsyntax (sh) 216
 - Begriffe 9
 - Bezeichner 9
 - Bourne- 6, 11
 - Bourne-Again- 585
 - C- 6, 477
 - Debuggen (csh) 539
 - Eigenschaften 7
 - Eigenschaften der 7
 - Einfache Kommandos 13
 - Fehlersuche (csh) 539
 - Fehlersuche (sh) 168
 - Funktion (sh) 154
 - Funktionen (bash) 601
 - Funktionen (bash) 590
 - Funktionen (ksh) 332
 - Job-Kontrolle (sh) 229
 - Kommandozeile 9
 - Kommentar 12
 - Kommentar (csh) 480
 - Korn- 6, 249
 - Layers (sh) 230
 - Listen 13
 - Login- 7
 - Metazeichen 9
 - Metazeichen (csh) 479
 - Metazeichen (ksh) 253
 - Metazeichen (sh) 11
 - Optionen (bash) 588
 - Optionen (csh) 568
 - Optionen (ksh) 430
 - Optionen (sh) 206, 216
 - Parameter (bash) 591
 - Parameter (csh) 486
 - Parameter (ksh) 261
 - Parameter (sh) 31
 - PID (csh) 494
 - Pipeline 13
 - Potionen (tcsh) 639
 - Programmiersprachkonstrukte (bash) 589
 - Programmiersprachkonstrukte (csh) 517
 - Programmiersprachkonstrukte (ksh) 319
 - Programmiersprachkonstrukte (sh) 107
 - Prozeduren (csh) 485
 - Prozeduren (ksh) 257
 - Prozeduren (sh) 22
 - Prozeßnummer (csh) 494
 - Quoting (sh) 67
 - Restricted 227
 - rsh 227
 - shl 229
 - Signalbehandlung (csh) 540
 - Signalbehandlung (sh) 180
 - Skript (csh) 485
 - Skripts (csh) 485
 - Skripts (ksh) 257
 - Skripts (sh) 22
 - Sub- 12
 - Sub- (csh) 480, 485, 517
 - tcsh 639
 - Trennzeichen 9
 - Überlagern mit anderem Kommando (csh) 565
 - Variable
 - frei wählbar (csh) 488
 - Variablen (sh) 35

- Varianten 6
- Verlassen (csh) 565
- Virtuelle Terminals 230
- Wort 10
- SHELL (csh) 493
- SHELL (ksh) 252, 271
- SHELL (sh) 44
- SHELL (tcsh) 707, 720
- Shell-Skript
 - Aufruf (csh) 485
 - Debuggen (csh) 539
 - Debuggen (ksh) 385
 - Debuggen (sh) 168
 - Fehlersuche (csh) 539
 - Fehlersuche (ksh) 385
 - Signalbehandlung (csh) 540
 - Signalbehandlung (ksh) 387
 - Signalbehandlung (sh) 180
 - Signalbehandlung (tcsh) 679
- Shell-Variable
 - \$ (csh) 494
 - { } (ksh) 282
 - * (csh) 494
 - addsuffix (tcsh) 708
 - AFSUSER (tcsh) 725
 - afsuser (tcsh) 708
 - allow_null_glob_expansion (bash) 594
 - ampm (tcsh) 708
 - argv (csh) 494, 495
 - argv (tcsh) 708
 - Arrays (csh) 489, 498
 - Arrays (ksh) 265, 281, 284
 - Attribute (ksh) 276
 - Ausdrücke (bash) 597, 598
 - Ausdrücke (csh) 497
 - Ausdrücke (ksh) 282
 - Ausdrücke (sh) 47
 - Ausdrücke (tcsh) 665
 - Ausgeben (alle) 45
 - Ausgeben (sh) 61
 - autocorrect (tcsh) 646, 708
 - autoexpand (tcsh) 646, 708
 - autolist (tcsh) 708
 - autologout (tcsh) 709
 - automatische (csh) 494
 - backslash_quote (tcsh) 709
 - BASH (bash) 592
 - BASH_VERSION (bash) 592
 - cdable_vars (bash) 594
 - CDPATH (bash) 594
 - cdpath (csh) 491
 - CDPATH (ksh) 267
 - CDPATH (sh) 42
 - cdpath (tcsh) 709
 - cdspell (csh) 574
 - Child (csh) 495
 - child (csh) 495
 - COLUMNS (ksh) 267, 359
 - COLUMNS (tcsh) 725
 - command (tcsh) 709
 - command_oriented_history (bash) 594
 - complete (tcsh) 646, 709
 - correct (tcsh) 646, 710
 - cwd (csh) 495
 - cwd (tcsh) 710
 - dextract (tcsh) 710
 - dirsfile (tcsh) 669, 710
 - dirstack (tcsh) 669, 710
 - DISPLAY (tcsh) 725
 - dunique (tcsh) 710
 - echo (csh) 491, 540
 - echo (tcsh) 710
 - echo_style (tcsh) 711
 - edit (tcsh) 711
 - EDITOR (ksh) 267
 - EDITOR (tcsh) 725
 - ellipsis (tcsh) 711
 - ENV (bash) 594
 - ENV (ksh) 252, 267, 439
 - ERRNO (ksh) 273
 - EUID (bash) 592
 - Exportieren 59
 - Exportieren (sh) 61
 - FCEDIT (bash) 594
 - FCEDIT (ksh) 268
 - IGNORE (bash) 594
 - ignore (tcsh) 645, 711
 - filec (tcsh) 711
 - FPATH (ksh) 268, 342
 - gid (tcsh) 711
 - glob_dot_filenames (bash) 594
 - GROUP (tcsh) 725
 - group (tcsh) 711
 - Gültigkeitsbereich (sh) 56
 - histchars (bash) 594
 - histchars (csh) 491
 - histchars (tcsh) 711
 - HISTCMD (bash) 592
 - HISTCONTROL (bash) 594
 - histdup (tcsh) 712

- HISTFILE (bash) 594
- HISTFILE (ksh) 268
- histfile (tcsh) 712
- HISTFILESIZE (bash) 595
- histlit (tcsh) 712
- history (csh) 492, 529
- history (tcsh) 712
- history_control (bash) 594
- HISTSIZE (bash) 595
- HISTSIZE (ksh) 268
- HOME (bash) 595
- home (csh) 492
- HOME (ksh) 268
- HOME (sh) 41
- HOME (tcsh) 725
- home (tcsh) 712
- HOST (tcsh) 725
- HOSTFILE (bash) 595
- hostname_completion_file (bash) 595
- HOSTTYPE (bash) 592
- HOSTTYPE (tcsh) 725
- HPATH (tcsh) 725
- IFS (bash) 595
- IFS (ksh) 268
- IFS (sh) 44
- IGNOREEOF (bash) 595
- ignoreeof (csh) 492
- ignoreeof (tcsh) 713
- inputmode (tcsh) 713
- INPUTRC (bash) 595
- LANG (tcsh) 725
- LC_TYPE (tcsh) 725
- LINENO (bash) 593
- LINENO (ksh) 273
- LINES (ksh) 269
- LINES (tcsh) 725
- listflags (tcsh) 713
- listjobs (tcsh) 677, 713
- listlinks (tcsh) 713
- listmax (tcsh) 647, 714
- listmaxrows (tcsh) 647, 714
- loginsh (tcsh) 714
- LOGNAME (sh) 44
- logout (tcsh) 714
- Löschen 38
- Löschen (ksh) 265
- Löschen (sh) 61
- MACHTYPE (tcsh) 725
- MAIL (bash) 595
- mail (csh) 492
- MAIL (ksh) 269
- MAIL (sh) 42
- mail (tcsh) 714
- MAILCHECK (bash) 595
- MAILCHECK (ksh) 269
- MAILPATH (bash) 595
- MAILPATH (ksh) 269
- MAILPATH (sh) 42, 43
- MAILWARNING (bash) 595
- matchbeep (tcsh) 646, 714
- no_exit_on_failed_exec (bash) 595
- no beep (tcsh) 646, 715
- noclobber (bash) 595
- noclobber (csh) 492
- noclobber (tcsh) 715
- noglob (csh) 492, 502
- noglob (tcsh) 715
- nokanji (tcsh) 715
- nolinks (bash) 595
- nonomatch (csh) 493
- nonomatch (tcsh) 715
- NOREBIND (tcsh) 725
- nostat (tcsh) 715
- notify (bash) 595
- notify (csh) 493
- notify (tcsh) 716
- oid (tcsh) 716
- OLDPWD (bash) 593
- OLDPWD (ksh) 274
- OPTARG (bash) 593
- OPTARG (ksh) 274
- OPTERR (bash) 596
- OPTIND (bash) 593
- OPTIND (ksh) 274
- OSTTYPE (bash) 593
- OSTYPE (tcsh) 726
- owd (tcsh) 716
- PATH (bash) 596
- path (csh) 493
- PATH (ksh) 252, 269
- PATH (sh) 41
- PATH (tcsh) 726
- path (tcsh) 716
- PPID (bash) 593
- PPID (ksh) 274
- printexitvalue (tcsh) 716
- prompt (csh) 493, 529
- prompt (tcsh) 716
- PROMPT_COMMAND (bash) 593, 603
- prompt2 (tcsh) 718
- prompt3 (tcsh) 719
- promptchars (tcsh) 719

- PS1 (bash) 596, 603
- PS1 (ksh) 270
- PS1 (sh) 43
- PS2 (bash) 596
- PS2 (ksh) 270
- PS2 (sh) 43
- PS3 (bash) 596
- PS3 (ksh) 270, 326
- PS4 (bash) 596
- PS4 (ksh) 270
- pushdsilent (tcsh) 719
- pushdtohome (tcsh) 719
- PWD (bash) 593
- PWD (ksh) 274
- PWD (tcsh) 726
- RANDOM (bash) 593
- RANDOM (ksh) 274
- recexact (tcsh) 646, 719
- recognize_only_executables (tcsh) 647
- regognize_only_executables (tcsh) 719
- REMOTEHOST (tcsh) 726
- REPLY (bash) 593
- REPLY (ksh) 274, 327
- rmstar (tcsh) 719
- savedirs (tcsh) 669, 719
- savehist (csh) 493
- savehist (tcsh) 719
- sched (tcsh) 720
- SECONDS (bash) 593
- SECONDS (ksh) 274
- Setzen (csh) 487
- Setzen (ksh) 263
- Setzen (sh) 32
- SHACCT (ksh) 270
- SHACCT (sh) 44
- shell (csh) 493
- SHELL (ksh) 252, 271
- SHELL (sh) 44
- shell (tcsh) 720
- SHLVL (bash) 593
- SHLVL (tcsh) 726
- shlvl (tcsh) 720
- showdots (tcsh) 720
- status (csh) 482, 495
- status (tcsh) 720
- symlinks (tcsh) 720
- SYSTYPE (tcsh) 726
- tcsh (tcsh) 721
- term (csh) 493
- TERM (ksh) 271, 359
- TERM (sh) 44
- TERM (tcsh) 726
- term (tcsh) 721
- TERMCAP (tcsh) 726
- time (csh) 493
- time (tcsh) 722
- TMOUT (bash) 596
- TMOUT (ksh) 271
- tperiod (tcsh) 722
- tty (tcsh) 722
- TZ (ksh) 271
- TZ (sh) 44
- UID (bash) 593
- uid (tcsh) 723
- USER (tcsh) 726
- user (tcsh) 723
- VENDOR (tcsh) 726
- verbose (csh) 494, 540
- verbose (tcsh) 723
- version (tcsh) 723
- visiblebell (tcsh) 724
- VISUAL (ksh) 271
- VISUAL (tcsh) 726
- vordefiniert und änderbar (csh) 491
- vordefinierte (csh) 491
- watch (tcsh) 724
- Wert 12
- Wert (csh) 480, 486, 488
- Wert-Zuweisung 12
- who (tcsh) 724
- wordchars (tcsh) 725
- Shell-Variable (csh) 487
- Shell-Variable (ksh) 264
- shift (bash) 626
- shift (csh) 487, 559, 566
- shift (ksh) 263, 415
- shift (sh) 34, 210
- shift (tcsh) 702
- shl 229
- SHLVL (bash) 593
- SHLVL (tcsh) 726
- shlvl (tcsh) 720
- showdots (tcsh) 720
- Signalbehandlung (csh) 540
- Signalbehandlung (ksh) 387
- Signalbehandlung (sh) 180
- Signalbehandlung (tcsh) 679
- Signale (csh) 540
- Skript
 - achtdame (csh) 549
 - ausführbar machen 485
 - Automatisieren von Umfragen (csh) 575

- Beispiel für Job-Kontrolle (csh) 549
- Beispiel zu goto (csh) 528
- Beispiel zu Ko-Prozeß 254
- Beispiel zu onintr (csh) 541
- copy (csh) 519
- csh 485
- Dateien in Directories zählen (csh) 518
- delfang (csh) 541
- Directorybaum graphisch ausgeben (csh) 526
- hexziff (csh) 521
- hexziff2 (csh) 522
- ksh 257
- loesch (csh) 521
- Löschen mit Sicherungskopie (csh) 521
- Löschen von Prozessen über Namen(csh) 527
- MSDOS-copy (csh) 519
- namkill (csh) 527
- rechne 254
- sh 22
- tree (csh) 526
- userda (csh) 524
- wahl (csh) 575
- wahl_info (csh) 575
- wcdat (csh) 518
- wzaehl (csh) 528
- source (bash) 626
- source (csh) 560, 566
- source (tcsh) 703
- Stack
 - Directory- (csh) 553
- Standardausgabe
 - umlenken 11
 - Umlenken (csh) 506
 - umlenken (csh) 479, 481
 - umlenken (ksh) 253
 - Umlenken (noclobber) 506
 - umlenken ans Dateiende 11
 - Umlenken ans Dateiende (csh) 507
 - umlenken ans Dateiende (csh) 479
- Standardausgabe (csh) 506
- Standardeingabe
 - umlenken 11
 - Umlenken (csh) 506
 - umlenken (csh) 479
- Standardeingabe (csh) 506
- Standardfehlerausgabe
 - Umlenken (csh) 506
 - Umlenken (noclobber) 506
 - Umlenken ans Dateiende (csh) 507
 - Standardfehlerausgabe (csh) 506
 - Starten der csh 478
 - Starten der ksh 251
 - status (csh) 482, 495
 - status (tcsh) 720
 - stderr (csh) 506
 - stdin (csh) 506
 - stdout (csh) 506
 - stop (csh) 548, 566
 - stop (ksh) 416
 - stop (tcsh) 703
 - Subshell 12
 - Subshell (csh) 480, 485, 517
 - Substitution
 - Alias- (csh) 537
 - History- 530
 - Kommando- 12
 - Kommando- (csh) 480, 485
 - Parameter- (csh) 497
 - Suchpfade
 - cd (csh) 491
 - Programme (csh) 493
 - suspend (bash) 627
 - suspend (csh) 549, 566
 - suspend (ksh) 416
 - suspend (tcsh) 703
 - switch (csh) 520
 - switch (tcsh) 703
 - switch-Anweisung (csh) 520
 - symlinks (tcsh) 720
 - Syntaxdiagramme zu Pipeline und Kommandoliste (ksh) 256
 - SYSTYPE (tcsh) 726

T

 - Tabulatorzeichen 12
 - Tabulatorzeichen (csh) 480
 - Tcsh
 - Signalbehandlung 679
 - tcsh 639
 - .cshrc 640
 - .login 641
 - .tcshrc 640
 - Alias-Mechanismus 664
 - Anwendungsbeispiele 726
 - Aufrufsyntax 639
 - Ausdrücke 673

- Builtin-Editoren 641
- Einfache Kommandos 670
- History-Mechanismus 659
- Job-Kontrolle 677
- Listen 670
- Optionen 639
- Pipeline 670
- tcsh (tcsh) 721
- telltc (tcsh) 704
- TERM (csh) 493
- TERM (ksh) 271, 359
- TERM (sh) 44
- TERM (tcsh) 721, 726
- TERMCAP (tcsh) 726
- test (bash) 627
- test (ksh) 306, 415
- test (sh) 92, 210
- Text ausgeben (csh) 564
- Tilde (bash) 597
- Tilde (csh) 481, 502
- Tilde (ksh) 253, 353
- Tilde (tcsh) 668
- Tilde-Expandierung (bash) 597
- Tilde-Expandierung (ksh) 250, 353
- time (csh) 493, 561, 566
- time (ksh) 331
- time (tcsh) 704, 722
- times (bash) 629
- times (ksh) 411, 415
- times (sh) 209, 211
- Tischrechner
 - bc 729, 738
 - dc 729
- TMOUT (bash) 596
- TMOUT (ksh) 271
- tperiod (tcsh) 722
- Tracked Aliase (ksh) 347
- trap (bash) 629
- trap (ksh) 389, 415
- trap (sh) 182, 211
- tree (csh-Skript) 526
- Trennzeichen 9
- true (ksh) 349, 416
- tty (tcsh) 722
- type (bash) 630
- type (ksh) 349, 416
- type (sh) 164, 211
- typeset (bash) 631
- typeset (ksh) 276, 415
- TZ (ksh) 271
- TZ (sh) 44

U

- überlange Kommandozeilen 219
- Überlagern der Shell mit anderem
 - Kommando (csh) 565
- UID (bash) 593
- uid (tcsh) 723
- ulimit (bash) 631
- ulimit (ksh) 412, 415
- ulimit (sh) 209, 211
- umask (bash) 632
- umask (csh) 566
- umask (ksh) 415
- umask (sh) 211
- umask (tcsh) 704
- Umgebung
 - ksh 428
- Umlenken
 - Hier-Dokument (csh) 507
 - Standardausgabe 11
 - Standardausgabe (ans Dateieende)
 - (csh) 507
 - Standardausgabe (csh) 479, 481, 506
 - Standardausgabe (ksh) 253
 - Standardausgabe (noclobber) 506
 - Standardeingabe 11
 - Standardeingabe (csh) 479, 506
 - Standardfehlerausgabe (ans Dateieende)
 - (csh) 507
 - Standardfehlerausgabe (csh) 506
 - Standardfehlerausgabe (noclobber) 506
- Umlenken (csh) 506
- Umlenkung
 - Ausgabe (bash) 600
 - Ausgabe (csh) 506
 - Ausgabe (ksh) 293
 - Ausgabe (sh) 78
 - Ausgabe (tcsh) 672
 - Eingabe (bash) 600
 - Eingabe (csh) 506
 - Eingabe (ksh) 293
 - Eingabe (sh) 78
 - Eingabe (tcsh) 672
 - Operatoren (csh) 506
- unalias (bash) 632
- unalias (csh) 539, 566
- unalias (ksh) 346, 415
- unalias (tcsh) 704
- uncomplete (tcsh) 705
- unhash (csh) 557, 566
- unhash (tcsh) 705

unlimit (csh) 564, 566
 unlimit (tcsh) 705
 unset (bash) 632
 unset (csh) 566
 unset (ksh) 265, 341, 415
 unset (sh) 38, 61, 211
 unset (tcsh) 705
 unsetenv (csh) 558, 566
 unsetenv (tcsh) 705
 until (bash) 590
 until (ksh) 324
 until (sh) 135
 user (tcsh) 723, 726
 userda (csh-Skript) 524

V

Variable
 Shell- (csh) 487
 Variablen
 arithmetisch (ksh) 310
 Variablen (csh) 487
 Variablen (ksh) 264
 Variablen (sh) 35
 Variablenausdrücke (bash) 597, 598
 Variablenausdrücke (csh) 497
 Variablenausdrücke (ksh) 282
 Variablenausdrücke (sh) 47
 Variablenausdrücke (tcsh) 665
 VENDOR (tcsh) 726
 ver (tcsh) 705
 verbose (csh) 494, 540
 verbose (tcsh) 723
 Verlassen der Shell (csh) 565
 Verschieben der Positionsparameter
 (csh) 487
 version (tcsh) 723
 Verzweigung (csh) 517, 520
 vi (ksh) 371
 visiblebell (tcsh) 724
 VISUAL (ksh) 271
 VISUAL (tcsh) 726
 Vordefinierte Aliase (ksh) 349, 416
 Vordefinierte Shell-Variablen
 änderbar (csh) 491
 Vordefinierte Shell-Variablen (csh) 491
 Vordergrund-Kommando
 Exit-Status (csh) 495

W

wahl (csh-Skript) 575
 wahl_info (csh-Skript) 575
 wait (bash) 632
 wait (csh) 547, 566
 wait (ksh) 396, 415
 wait (sh) 211
 wait (tcsh) 705
 watch (tcsh) 724
 watchlog (tcsh) 706
 wcdat (csh-Skript) 518
 Wert einer Shell-Variablen (csh) 488
 whence (ksh) 402, 415
 where (tcsh) 706
 which (tcsh) 706
 while (bash) 590
 while (csh) 523
 while (ksh) 323
 while (sh) 129
 while (tcsh) 706
 while-Anweisung (csh) 523
 who (tcsh) 724
 wordchars (tcsh) 725
 Working-Directory
 495
 Wort 10
 wzaehl (csh-Skript) 528

X

xargs 220

Z

Zeichen
 Leer- 12
 Leer- (csh) 480
 Neuezeile- 12
 Neuezeile- (csh) 480
 Tabulator- 12
 Tabulator- (csh) 480
 Zweiwege-Pipe 253